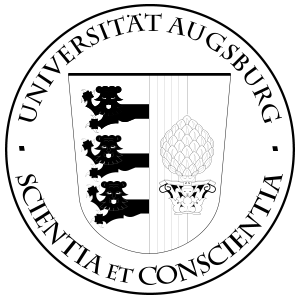
Relational and Algebraic Calculi for Database Preferences

Dissertation zur Erlangung des Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat)

an der Fakultät für Angewandte Informatik
der Universität Augsburg



vorgelegt von

Patrick Roocks

2016

Referees:
Prof. Dr. Bernhard Möller
Prof. Dr. Werner Kießling

Day of Defense:
May 3, 2016

Abstract

Relational methods in computer science have been studied intensively in the last decades, especially for program verification and correctness. In the present thesis we apply them to database preferences, which are a generalization of Skyline queries. This topic is connected to relations and algebra in various respects. The formal basis of databases is given by the relational data model, and preferences are strict order relations on tuples from a given data set. Moreover, preference operators and operands form an algebraic structure by themselves, which led to the research field of algebraic optimization of database preference queries.

In this work we develop a coherent family of calculi for dealing with database preferences. Wherever possible, we use algebraic structures such as semirings, abstract relation algebras, and related concepts. The relational algebraic approach allows us to reason about many aspects of Skyline computation and preference term equivalences in a point-free way. We generalize and unify existing theorems in the scope of database preferences and simplify some of their proofs by means of algebraic structures. Next to this, we introduce the new field of preference decomposition. A subgoal of this is the characterization of the expressiveness of preference queries, i.e., the classification of orders constructed by a given class of preference terms. The results of this thesis have various applications regarding the correctness, soundness and efficiency of database preference implementations.

In addition to our theoretical contributions we implemented the *rPref* package (available at CRAN) for handling preferences within the statistical computing software R. There is a tight connection between our calculi and the query language in that package. This allowed us to implement the algorithms and examples from the theoretical parts of this thesis, demonstrating the applicability of our results.

Acknowledgement

I want to thank all people who supported and motivated me during the work on this thesis.

First of all, I am very grateful to my supervisor, Prof. Dr. Bernhard Möller, University of Augsburg, for his support, encouragement and the great cooperation over all the years. He drew my attention to the problems considered in this thesis, carefully read all my papers including many drafts of them, and gave very detailed, valuable and helpful feedback.

Next, I am grateful to Prof. Dr. Werner Kießling, University of Augsburg, who provided my research position for the last 5 years and supported me in the applied aspects of my thesis. The possibility to work in several industrial-academic research projects guided by him and the opportunity to visit many international database conferences gave me valuable input and motivation for my work. Further thanks for the financial support of these projects to the Bavarian Ministry of Economic Affairs and Media, Energy and Technology.

I am grateful to my (former) colleagues Han-Hing Dang, Markus Endres, Roland Glück, Alfons Huhn, Stefan Mandl (now at Exasol AG), Martin Müller, Lena Rudenko, Florian Wenzel, and Andreas Zelend for the good atmosphere at our chair. In particular, thanks to Markus Endres, Stefan Mandl, and Florian Wenzel for investing that much time in writing the research grant proposals funding my position. Thanks to Alfons Huhn for drawing my attention to the preference decomposition problem.

It was a great motivation for me to present and discuss my work on the RAMiCS (Relational and Algebraic Methods in Computer Science) and MPC (Mathematics of Program Construction) meetings. I am grateful for Bernhard Möller's encouragement and both mental and financial support to participate there. I want to thank all the anonymous referees of these conference series for their valuable comments.

Special thanks to my girlfriend Carla for her love, support, and understanding in the last 2.5 years of my work and for proofreading this thesis. Finally, I would like to thank all my friends who supported me through the last years. In particular, thanks to Christoph Gietl and Jan Natolski from the department of mathematics for the interesting discussions during lunch.

Contents

Chapter 1

Introduction

Database preferences allow expressing soft constraints within database queries. In this chapter we present an overview concerning their historical development from the Pareto principle in economics to the recent developments of preference implementations in database systems. The work on them, in particular the optimization and efficient evaluation of preference queries, poses many theoretical challenges. We motivate our approach to these problems based on relational and algebraic methods. Additionally, we sketch how our formal preference framework is connected to the implementations. Finally we will summarize the structure of this thesis.

1.1 The History of Database Preferences

Database preferences have been one of the major new developments in the area of databases in the last decades. The basic idea is the simple principle of *Pareto optima*, well known in economics and named after the engineer, economist and sociologist *Vilfredo Pareto*. An object is said to be *Pareto optimal* if there is no other object, which is *dominating* this object. Here and in the following, the term *dominating* means that one object is better or equal in all dimensions of interest and strictly better in at least one dimension.

We assume that at least two dimensions are considered. As the computational costs to calculate the Pareto optima quickly increase with the number of relevant dimensions, this number is typically "low". For the most use cases in the scope of large databases, it normally ranges from 2 to 5.

Usually, this principle is applied to data sets whose dimensions tend to be conflicting. For example, consider Figure 1.1, where the data set consists of hotels in a coastal region. We search for hotels which are near to the beach *and* cheap. The Pareto optimal set of hotels is visualized by a continuous line the Figure. As it can be seen from the picture, these dimensions are weakly anticorrelated. This means, the price tends to decrease with a growing distance from the beach, as guests are willing to pay more for rooms with a nice view and short way to the beach.

The pioneering work of introducing this concept to the area of databases was done in [BKS01] using the *Skyline operator*. This name stems from the fact that the Pareto front line in the diagram are those points which are visible when viewed from the hypothetical optimum, i.e., the point $(0, 0)$ in Figure 1.1.

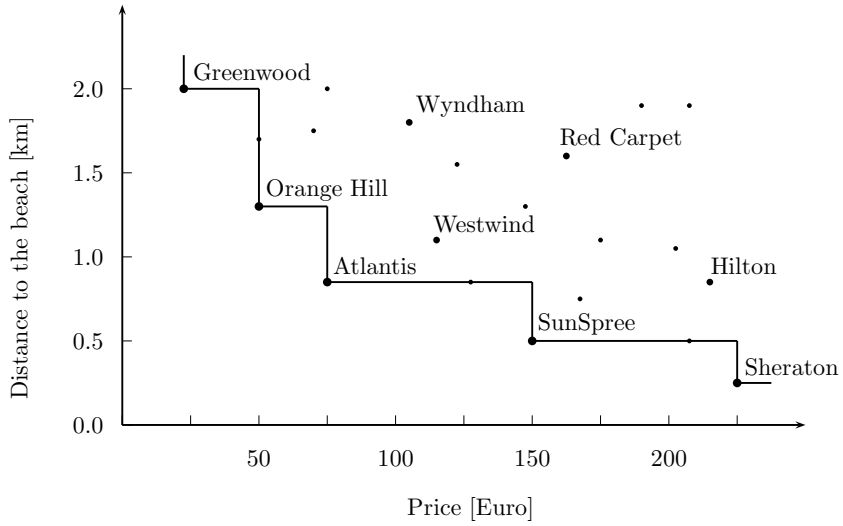


Figure 1.1: Pareto optimal set of hotels, where the dimensions *price* and *distance to the beach* are simultaneously optimized.

Generalizations to *database preferences* with an extended set of operators have been introduced in [Kie02, Kie05, Cho03] and many following papers, many building on those three papers. In these works, additional constructs are introduced into the preference framework. A nearby generalization consists in preference top-k queries, where the k best tuples w.r.t. the preference order are returned. As another example, in [Kie02], lexicographic orders can be combined with the Pareto principle. In the same work a neat semantic definition of a preference query language is suggested, and a variety of rules for simplification of the constructs and optimization of the query evaluation are given. A first comprehensive implementation of this preference framework has been realised in the research prototype *Preference SQL* [KK02].

Within the last decade a plenty of sophisticated algorithms for processing preference queries and semantic extensions for the concept have been suggested. Mostly we will not go into their details, as this would be beyond the scope of this thesis.

1.2 Our Approach

In this thesis we mainly focus on the theoretical aspects of preference, relying on the framework introduced in [Kie02]. Especially we consider the Pareto operator, leading to the Skylines described above, in connection with the *prioritization* operator, modelling lexicographic orders. The main challenge of our work was to find concise and elegant ways to prove theoretical properties of database preferences and associated methods, which are of interest for implementing a preference query language.

The development of preference query languages and the study of the use cases from the application posed a lot of theoretical challenges. Primarily the theory we will present underlines the *soundness* of the concept, i.e., that all the preference constructs have precise formal definitions and work well together. For example, all preference operators preserve strict orders, i.e., the soundness of a preference term can be derived from its syntactical correctness.

A main challenge of the integration of preferences into databases are the optimization issues. The theorems concerning the simplification of preference terms and an optimized

query processing are very important for the applications. Another theoretically interesting problem is to characterize the expressiveness of the preference query languages supported by the current available implementations. This means, we want to characterize which set of preference constructs is necessary to express different kinds of orders (total orders, weak orders, etc.).

To show the validity of these theoretical results, we strive for a close connection to different implementations. Starting from the pioneering works on the research prototype *Preference SQL* [KK02, KEW11], two implementations have emerged in the last two years, in which we were involved. One of them is our freely available *rPref* package [Roo15f] for the *statistical computing language R* [R C15]. This work initially began with the development of a rapid prototyping framework for preferences [RK13]. A syntactically similar preference language is contained in the commercially available database system *Exasol Exasolution* with the *Skyline* feature [MKEK15]. This one was developed in an academic-industrial cooperation between the chair of databases at the university of Augsburg and the Exasol company.

To keep a tight connection between theory and practice, we will motivate and exemplify our theoretical results with the preference query languages used in *rPref* and *Exasolution Skyline*, which are quite similar. Beyond that, we will point out where the experiences from the practice have influenced the development of the tools.

1.3 Relational and Algebraic Methods

The goal of this thesis is to study the theoretical problems described above using well-established algebraic concepts. With this, we get a more abstract view of the problem and can point out connections to relational approaches for other problems. Especially we can reuse many results around abstract relation algebras. Based on this theory, we suggest a point-free and lean formalism for database preferences. Many theorems which were originally formulated as quantified implications can be shown in a purely equational, and thus more readable, fashion.

The most important theoretical foundations for our work are the theories around semirings and Kleene algebra [DMS06] and abstract relation algebras [SS93, Mad97]. These algebraic axiomatizations pave the way for the use of automated theorem provers like Prover9, cf. [McC05]. For Kleene algebra, there exist well suited axiomatizations for these tools, cf. [HS08]. This approach allows studying precisely which assumptions, i.e., axioms, are necessary to derive the properties which we want show. Thus, we also obtain a deeper understanding of the theory.

Based on these structures we introduce the *join algebra*, which is an extension of an abstract relation algebra. This allows expressing the most important preference operators, Pareto and Prioritization, in a point-free fashion. Most proofs in this thesis fundamentally rely on this idea.

1.4 Contribution and Organisation

This thesis is divided into two main parts: Chapter 2 and 3 introduce the theory around database preferences and relational and algebraic calculi. This is the more foundational part of the thesis. The second part, Chapters 4–6, is dedicated to the applications of preferences. We will study some problems motivated by practice and describe an implementation of database preferences. Additionally we discuss how this implementation

and tools like automated theorem provers allowed comprehending some of the theoretical results of this thesis.

In the following we summarize the main contributions of each chapter and reference the related publications, where most results of this thesis have been published.

- In Chapter 2 we model data sets and preferences as typed sets and relations following [MRE12, MR15]. We introduce the preference framework from [Kie02, Kie05] and adapt it to our formalism. This chapter defines a concrete relational model of database preferences. Beyond this, we point out how this model is related to query languages.

- In Chapter 3 we introduce the most important algebraic structure for reasoning about database preferences in a more abstract way. The central concept is the *join algebra*, introduced in different variants in [MRE12, MR12b, MR15]. We generalize and extend the definition of preferences from the second chapter to fit in the algebraic approach. We show results from [MRE12, MR15] regarding the theory of processing preference queries, i.e., transformations for calculating maximal objects for a preference and equivalences of preference terms. Additionally, we show an algebraic approach to the special case of *layered preferences* from [MR12b, MR15] and derive new constructs for preference query languages based on this.

- In Chapter 4 we present some problems from practice. For example, we show how our algebraic approach simplifies the proofs for transformations of grouped preference queries, published in [ERK14]. We give an algebraic derivation of an optimization for the *Online Skyline problem*, where preferences are applied to a stream of data points, presented in one chapter of [DGM+14]. Finally, we treat the Scalagon algorithm [ERK15] for processing Pareto preferences, which is a concrete example of a prefilter.

- In Chapter 5 we study the expressiveness of common preference query languages by defining a decomposition of every preference into a restricted set of preference constructs [Roo15c]. We see that different choices of operators and operands result in different kinds of resulting orders, ranging from total orders to strict orders. Additionally, we present an optimized decomposition and apply this method to preferences on power sets [Roo15a]. Finally, we study the term complexity of the decomposition approaches.

- In Chapter 6 we show different approaches of implementing the theory and give some practical evidence of our results. For a selected theorem from Chapter 4, we show a proof script for an automated theorem proofer. Next, we describe the rPref package [Roo15f], that is based on the statistical programming language R. This follows the idea of rapid prototyping database preferences as presented in [RK13], and bundles all the functionality in one package, available at the official R package repository CRAN. We will show some use cases, where methods from the theoretical part of this thesis are implemented and evaluated within rPref.

We end with a conclusion and an outlook, containing a summary of our results, a critical reflection of our work and list some open questions for future research.

In the appendix, we show selected R scripts, implementing some of the theoretical constructs related to the preference decomposition, as presented in Chapter 5. These scripts are developed on top of the rPref package.

CHAPTER 2

A Relational Approach

Both the relational model of databases as well as our model of preferences in the sense of strict partial orders are instances of relational algebra. We build a formal framework covering database relations and preference relations in a formally consistent way. An important component of our approach is a typing mechanism for relations, establishing a connection between attributes of a data set and preference terms. We introduce operators and operands defining a language of preference terms inducing strict partial orders and point out some connections between theory and implementations of preference query languages. These languages especially support the Skyline operator. The introduction of point-free representations of preference operators pave the way for a generalised view on the topic.

2.1 Types, Tuples and Typed Relations

The pioneering work in relational databases has been done with the definition of the *relational data model* [Cod70]. The first relational database management systems, especially IBM's *System R* and *Oracle*, were inspired by Codd's work. In these early implementations, a query language called *SEQUEL* for relational databases was developed, which later on lead to *SQL* (*Structured Query Language*). More than one decade later SQL became an official standard. Up to now several revisions of this standard were released and nowadays SQL is a widely used query language for relational databases.

We strive to keep a tight connection between theory and practice throughout this work. In the specification of the formal framework we will frequently refer to the concepts that are already common practice in relational databases. Types and tuples, as we define them in the following, are the formal counterparts to database attributes with given data types. Tuples are rows of a relational query result.

Unfortunately, there is a major gap between theory and practice when comparing the relational theory and the SQL standard: Any relational query result is a *set of tuples* in the model of [Cod70] where duplicates cannot occur and the order does not matter. But in the definition of SQL, relational queries return in fact *lists of tuples*. In practice, the order of the elements matters and is commonly manipulated by sorting the elements with appropriate SQL directives.

Such differences led to a lots of criticism on SQL-based databases. For example, [DD95]

suggest to "get away from SQL and back to our relational roots" in their *"Third manifesto"*. But these differences do not affect our work at all. For all results of relational queries throughout this work, the order of the tuples does not matter. Whenever we establish connections between the relational calculus and the various implementations of database preferences, we will make appropriate assumptions to ensure that the theoretical and practical results are identical. For example, in order to avoid duplicates, we will require uniqueness of attribute values, if it is necessary.

The formalism of types, tuples and typed relations which we will present in the following is notationally and conceptually adapted to [Kan89]. This contains a formally more precise description of the relational model, compared to Codd's pioneering work. We will see in the next chapter that such a relational calculus is a good initial point to develop a more generalised algebraic calculus.

When introducing these concepts we will make use of a *point-free* manner. The idea behind this is to reduce the definitions of relational objects and operations to their essence. In the relational algebraic approach we try to replace any predicate containing universal or existential quantifier by formulas without them. The point-wise operations making use of quantifiers will be shifted back to the definition of fundamental operations, e.g., the image or domain of a relation. Their algebraic axiomatizations will be in the focus of the next chapter. By the use of point-free operations in relational formulas we strive for more elegant and more readable proofs. Instead of implication chains of quantified predicates, we will mostly deal with (in)equations of relational formulas.

2.1.1 Naming Conventions

To avoid misunderstandings we first introduce some naming conventions. The term *relation* has different meanings in the context of database preferences. It can refer to either a database relation or to a preference relation, i.e., a strict partial order. To avoid these ambiguity we introduce the following informal naming conventions.

- We will use the term *data set* for any data table, having named columns with associated data types.
- A *tuple* is one row of such a data set, having the same attributes as the data set.
- The term *set of tuples* is used for subsets of data sets or (intermediate) results of a query evaluation.
- A *relation* will be a homogeneous binary relation between tuples. In Section 2.2 we will specialize this concept to preferences and their associated equivalence relations.
- The concept of a *type* primarily establishes a connection between some type literal and a set of attributes (or columns) of a data set. In most cases the corresponding SQL data type (int, float, string, ...) does not matter and will be kept general or obtainable from the context.

2.1.2 Typed Tuples

Based on [Kan89] we define a precise formalism of types and tuples, informally introduced above. In the following definitions we mainly follow [MRE12].

Definition 2.1.1 (attributes, types and tuples)**.** Let $\mathcal{A}$ be a set of *attribute names* and a family $(D_A)_{A \in \mathcal{A}}$ of sets, where for $A \in \mathcal{A}$ the set D_A is called the *domain* of A. We define the following notions:

1. A *type* T is a subset $T \subseteq \mathcal{A}$.
2. An attribute $A \in \mathcal{A}$ is also used to denote the *singleton type* $\{A\}$, omitting the set braces. By convention, we use $A, B, C, ...$ for singleton types. For $|T| \geq 2$ we call T also a *complex type*.
3. A T-*tuple* is a mapping
$$t : T \to \bigcup_{A \in \mathcal{A}} D_A \text{ where } \forall A \in T : t(A) \in D_A \ .$$
4. For a T-tuple t and a sub-type $T' \subseteq T$ we define the projection $\pi_{T'}(t)$ to T' as the restriction of the mapping t to T':
$$\pi_{T'}(t) : T' \to \bigcup_{A \in \mathcal{A}} D_A \text{ with } A \mapsto t(A) \ .$$
The projection is extended to pairs of T-tuples by $\pi_{T'}((t_1, t_2)) = (\pi_{T'}(t_1), \pi_{T'}(t_2))$.
5. The domain D_T for a non-singleton type T is the set of all possible T-tuples, i.e., $D_T = \prod_{A \in T} D_A$.
6. The set $\mathcal{U} =_{df} \bigcup_{T \subseteq \mathcal{A}} D_T$ containing the domains of all possible types is called the *universe*.
7. For a tuple t, and a set of tuples M we introduce the following abbreviations:
$$t :: T \Leftrightarrow_{df} t \in D_T \ , \quad M :: T \Leftrightarrow_{df} M \subseteq D_T \ .$$
For $t :: T$ we say that t *is of type* T.

The domain of a singleton type D_A is typically $\mathbb{R}, \mathbb{N}$ (corresponding to `float`, `int` in the database context) or the set of all strings over a given alphabet (e.g., `varchar`, `text`).

As a fundamental operation we define a join operation on types and a corresponding operation on tuples. In the database context the join is widely known for merging data sets sharing some attributes. Typically two data sets being joined have one shared attribute which is, e.g., a unique identifier in one of the join partners and a foreign key among the other join partner. Such a setting is typically used for the relational modelling of $1 : n$ relations.

We primarily introduce the subsequent join operation to build complex types and complex typed set of tuples.

Definition 2.1.2 (join)**.** The *join* of two types T_1, T_2 is the union of their attributes:
$$T_1 \bowtie T_2 =_{df} T_1 \cup T_2 \ .$$

For sets of tuples $M_i :: T_i$ with $i = 1, 2$ the join is defined as the set of all consistent combinations of M_i-tuples:
$$M_1 \bowtie M_2 =_{df} \{t :: T_1 \bowtie T_2 \mid \pi_{T_i}(t) \in M_i, i = 1, 2\} \ .$$

Semantically the join operation on tuples is equivalent to the *natural join* in SQL. If two sets $M_1 :: T_1$ and $M_2 :: T_2$ have no attributes in common (formally $T_1 \cap T_2 = \varnothing$) the join $M_1 \bowtie M_2$ is equivalent to the Cartesian product $M_1 \times M_2$. If they have attributes in common, then the set $T_1 \cap T_2$ contains the join attributes. We illustrate the join operator for a non-empty set $T_1 \cap T_2$ in the following example.

Example 2.1.3. Assume a database of cars with unique id's and further attributes for the make and horsepower. Hence the attribute names, i.e., types, are $id, power$ and $make$. The type domain of the two first ones is $\mathbb{N}$ and D_{make} is the set of strings. The tuples are written as explicit mappings. Assume the following sets:

$$M_1 =_{df} \{\{id \mapsto 1,\ make \mapsto \text{'BMW'}\},\ \{id \mapsto 3,\ make \mapsto \text{'Audi'}\}\}\ ,$$
$$M_2 =_{df} \{\{id \mapsto 2,\ power \mapsto 150\},\ \{id \mapsto 3,\ power \mapsto 180\}\}\ .$$

The sets have the types $M_1 :: id \bowtie make$ and $M_2 :: id \bowtie power$. Now we consider the join $M_1 \bowtie M_2 :: id \bowtie make \bowtie power$. We have $(id \bowtie make) \cap (id \bowtie power) = id$. The only tuple $t :: id \bowtie make \bowtie power$ which fulfils both $\pi_{T_1}(t) \in M_1$ and $\pi_{T_2}(t) \in M_2$ is the one with $t : id \mapsto 3$. Hence the join result is given by

$$M_1 \bowtie M_2 = \{\{id \mapsto 3,\ make \mapsto \text{'Audi'},\ power \mapsto 180\}\}\ .$$

We state some rules for manipulating join expressions, which are very similar to those which hold for Cartesian products. For joins of disjoint types, we even have an isomorphism with Cartesian products, which is formally given in Part 5 of the following corollary.

Corollary 2.1.4 (join properties)**.** *The following laws hold:*

1. $\bowtie$ *is associative and commutative and distributes over* $\cup$*.*
2. $\bowtie$ *preserves the inclusion order, i.e., for* $M, M' :: T$ *and* $N :: T'$ *we have*
$$M \subseteq M' \ \Rightarrow\ M \bowtie N \subseteq M' \bowtie N\ .$$
3. *Assume* $M_i, N_i :: T_i$ *with* $i = 1, 2$*. Then the following exchange laws hold:*
$$(M_1 \cap N_1) \bowtie (M_2 \cap N_2) = (M_1 \bowtie M_2) \cap (N_1 \bowtie N_2)\ , \tag{2.1}$$
$$(M_1 \bowtie M_2) \times (N_1 \bowtie N_2) = (M_1 \times N_1) \bowtie (M_2 \times N_2)\ . \tag{2.2}$$
4. *For* $M, N :: T$ *we have* $M \bowtie N = M \cap N$*. In particular, we have* $N \bowtie N = N$*.*
5. *For* $M_i :: T_i$ *with* $i = 1, 2$ *and disjoint* T_i*, i.e.,* $T_1 \cap T_2 = \varnothing$*, the join* $M =_{df} M_1 \bowtie M_2$ *is isomorphic to the Cartesian product of* M_1 *and* M_2*.*

Proof. Parts 1 and 2 follow directly from the definitions. Using the definition of join and the usual intersection of sets we show the first exchange law (2.1) as follows:

$$\begin{aligned}
& x \in (M_1 \cap N_1) \bowtie (M_2 \cap N_2) \\
\Leftrightarrow\ & \pi_{T_1}(x) \in M_1 \cap N_1 \wedge \pi_{T_2}(x) \in M_2 \cap N_2 \\
\Leftrightarrow\ & \pi_{T_1}(x) \in M_1 \wedge \pi_{T_1}(x) \in N_1 \wedge \pi_{T_2}(x) \in M_2 \wedge \pi_{T_2}(x) \in N_2 \\
\Leftrightarrow\ & x \in M_1 \bowtie M_2 \wedge x \in N_1 \bowtie N_2 \\
\Leftrightarrow\ & x \in (M_1 \bowtie M_2) \cap (N_1 \bowtie N_2)\ .
\end{aligned}$$

For the second exchange law (2.2) we calculate similarly

$$\begin{aligned}
& (x, y) \in (M_1 \bowtie M_2) \times (N_1 \bowtie N_2) \\
\Leftrightarrow\ & x \in (M_1 \bowtie M_2) \wedge y \in (N_1 \bowtie N_2) \\
\Leftrightarrow\ & \pi_{T_1}(x) \in M_1 \wedge \pi_{T_2}(x) \in M_2 \wedge \pi_{T_1}(y) \in N_1 \wedge \pi_{T_2}(y) \in N_2 \\
\Leftrightarrow\ & (\pi_{T_1}(x), \pi_{T_1}(y)) \in M_1 \times N_1 \wedge (\pi_{T_1}(x), \pi_{T_1}(y)) \in M_2 \times N_2 \\
\Leftrightarrow\ & \pi_{T_1}((x, y)) \in M_1 \times N_1 \wedge \pi_{T_2}((x, y)) \in M_2 \times N_2 \\
\Leftrightarrow\ & (x, y) \in (M_1 \times N_1) \bowtie (M_2 \times N_2)
\end{aligned}$$

Regarding Part 4 we have by the definition of join and the typing assumptions $t \in M \bowtie N \Leftrightarrow t \in M \wedge t \in N$.

To show Part 5, consider $x \in M$. The two join conditions $\pi_{T_i}(x) \in M_i$ are independent. Hence all elements of M_1 can be joined with all elements of M_2. Thus, by definition,

$$t \in M \Leftrightarrow \pi_{T_1}(t) \in M_1 \wedge \pi_{T_2}(t) \in M_2 \Leftrightarrow (\pi_{T_1}(t), \pi_{T_2}(t)) \in M_1 \times M_2 \ . \qquad \square$$

Note that the second exchange law in Part 3 of the above corollary paves the way to defining binary relations over joined data sets. We will introduce typed binary relations in the next section.

A similar approach to formalizing typed data sets is given in [MO04]. There a calculus of typed database relations (i.e., data sets in our wording) is developed, especially suited for deduction systems. In contrast to our approach they define a disjoint union with implicit renaming. The disjoint union of typed sets having some attributes in common is soundly defined using an implicit renaming of those attribute names which occur in both typed sets. In our calculus we will assume that renaming of types is not necessary.

A further assumption is that we deal with *type-compatible* operations. The union of $M :: T$ and $N :: T'$ is type-compatible if and only if $T = T'$, i.e., the types are identical. In [Roo12] we have extended our calculus to unions of different types, called a *multityped* setting. There a specialized union operator "$T \cup_{\mathrm{m}} T'$" returns T if $T = T'$, i.e., if both types are identical. This expression returns the set of types, also called a *multitype*, $\{T, T'\}$ if they are different. We studied some theoretical properties in that short paper. As we have not yet found any application for multitypes we will restrict ourselves to type-compatible operations and will not go into the details of multitypes in this thesis.

2.1.3 Typed Relations

Up to now we have a notion of typed data sets and the join operator on sets and types. Using the usual definitions of set operations like union, intersection, difference and the projection as given in Definition 2.1.1 we can express some restricted set of relational queries. Compared to SQL, the selection operator (i.e., the `where` clause in a query), returning a subset of tuples fulfilling a given predicate, is not relevant for the most parts of this thesis. Instead we focus on an operator to find the optimal tuples w.r.t. a preference. In order to formally define this, we introduce homogeneous binary relations on typed tuples. They will serve as the formal basis to defining preferences on tuples.

Definition 2.1.5 (typed homogeneous binary relations)**.** For a type T we define the following abbreviations:

$$(t_1, t_2) :: T^2 \ \Leftrightarrow_{df} \ t_1, t_2 \in D_T \ , \qquad R :: T^2 \ \Leftrightarrow_{df} \ R \subseteq D_T \times D_T \ .$$

We say that the *typed relation* $R :: T^2$ has type T. There are some special relations for every type T: The full relation $\top_T =_{df} D_T \times D_T$, the identity $1_T =_{df} \{(x, x) \mid x \in D_T\}$ and the empty relation $0_T =_{df} \varnothing$.

Similar to the operations on data sets we define the join operation of such relations.

Definition 2.1.6 (join of relations)**.** Let $R_i :: T_i^2$ with $i = 1, 2$. Then the *join* $R_1 \bowtie R_2 :: (T_1 \bowtie T_2)^2$ is defined by

$$t \, (R_1 \bowtie R_2) \, u \Leftrightarrow_{df} \pi_{T_1}(t) \, R_1 \, \pi_{T_1}(u) \wedge \pi_{T_2}(t) \, R_2 \, \pi_{T_2}(u) \ .$$

For disjoint types T_1, T_2 (i.e., $T_1 \cap T_2 = \varnothing$) this is equivalent to the Cartesian product of relations. For joins of relations there hold the same laws as for joined data sets, cf. Corollary 2.1.4. Especially the algebraic properties (associativity, commutativity, distributivity over $\cup$ and preserving the inclusion order) of $\bowtie$ also apply to joined relations. We will extend two further important properties to joined relations in the subsequent corollary.

Corollary 2.1.7 (properties of joined relations)**.** *Assume $R_i, S_i :: T_i$ with $i = 1, 2$.*

1. *We have the exchange law*

$$(R_1 \cap S_1) \bowtie (R_2 \cap S_2) = (R_1 \bowtie R_2) \cap (S_1 \bowtie S_2) \ .$$

2. *For the join on the same type we get $R_1 \bowtie R_2 = R_1 \cap R_2$.*

Proof. Part 1 is analogous to the proof of (2.1) in Corollary 2.1.4.3. Part 2 is analogous to the proof of Corollary 2.1.4.4. □

Additionally the join operator distributes over the constructor for the special relations $0, 1$ and $\top$ as we state in following corollary.

Corollary 2.1.8 (properties of special relations on joins)**.** *For types T_1, T_2 and the placeholder $\mathsf{X} \in \{0, 1, \top\}$ for a typed binary relation we have $\mathsf{X}_{T_1 \bowtie T_2} = \mathsf{X}_{T_1} \bowtie \mathsf{X}_{T_2}$.*

Proof. Using the second exchange law (2.2) in Corollary 2.1.4 we infer $(D_{T_1} \bowtie D_{T_2}) \times (D_{T_1} \bowtie D_{T_2}) = (D_{T_1} \times D_{T_1}) \bowtie (D_{T_2} \times D_{T_2})$. By definition of the join for types we have that $T_1 \bowtie T_2 = T_1 \cup T_2$ holds. From the definition of the join for sets we infer that $D_{T_1 \bowtie T_2} = D_{T_1} \bowtie D_{T_2}$. This shows the claim for $\mathsf{X} = \top$. For $\mathsf{X} = 1$ we show the equality component-wise using again the argument $D_{T_1 \bowtie T_2} = D_{T_1} \bowtie D_{T_2}$. For $\mathsf{X} = \varnothing$ the claim is obvious. □

2.1.4 Inverse Image and Maximal Elements

Next to these fundamental properties on typed binary relations we introduce some basic operations: the inverse image and the maximum operator. They will give us a notion of optimal elements w.r.t. a strict partial order.

Definition 2.1.9 (inverse image)**.** For a relation $R :: T^2$ the inverse image of a set $Y :: T$ under R is formally defined as

$$|R\rangle Y \ =_{df} \ \{x :: T \mid \exists y \in Y : x \, R \, y\} \ .$$

The notation is borrowed from modal logic, where the inverse-image operator is a (forward) diamond. When considering the graph of a relation, the intuition behind the symbol is that $|R\rangle Y$ corresponds to those vertices having an R-edge pointing to a vertex in Y.

Lemma 2.1.10. *Assume $R_i :: T_i^2$ and $Y_i :: T_i$ with $i = 1, 2$ and disjoint T_1, T_2. Then the following exchange law for the join and the inverse image holds:*

$$|R_1 \bowtie R_2\rangle (Y_1 \bowtie Y_2) \ = \ |R_1\rangle Y_1 \bowtie |R_2\rangle Y_2 \ .$$

Proof. Using the definition of the inverse image and the join of relations we infer:

$$\begin{aligned}
& x \in |R_1 \bowtie R_2\rangle (Y_1 \bowtie Y_2) \\
\Leftrightarrow \ & \exists y \in (Y_1 \bowtie Y_2) : x \, (R_1 \bowtie R_2) \, y \\
\Leftrightarrow \ & \exists y \in (Y_1 \bowtie Y_2) : \pi_{T_1}(x) \, R_1 \, \pi_{T_1}(y) \ \wedge \ \pi_{T_2}(x) \, R_2 \, \pi_{T_2}(y) \\
\Leftrightarrow \ & \exists y_1 \in Y_1 : \exists y_2 \in Y_2 : \pi_{T_1}(x) \, R_1 \, y_1 \ \wedge \ \pi_{T_2}(x) \, R_2 \, y_2 \\
\Leftrightarrow \ & \pi_{T_1}(x) \in |R_1\rangle Y_1 \ \wedge \ \pi_{T_2}(x) \in |R_2\rangle Y_2 \\
\Leftrightarrow \ & x \in (|R_1\rangle Y_1 \bowtie |R_2\rangle Y_2) \ .
\end{aligned}$$

Note that splitting y into y_1 and y_2 in the third step is justified by disjointness of the types: because of $T_1 \cap T_2 = \varnothing$ the two join conditions $\pi_{T_i}(y) \in Y_i$ for $i = 1, 2$ are independent of each other, hence the substitution $y_i := \pi_{T_i}(y)$ is allowed. □

If R_1, R_2 are orders, i.e., transitive relations, and have disjoint types, a consequence of this lemma and Corollary 2.1.4.5 is that $R_1 \bowtie R_2$ is isomorphic to the *product order* $R_1 \times R_2$ on the Cartesian product domain $D_{T_1} \times D_{T_2}$.

The inverse image of a data set Y under a relation R, when viewed the other way around, consists of the tuples that have an R-successor in Y, i.e., are R-related to some object in Y. In the context of preferences (strict orders) we also say that $x\,R\,y$ means that x is *dominated* by y, hence $|R\rangle Y$ is the set of tuples dominated by Y. For this reason we can characterize the set of R-maximal objects within a set Y, as follows.

Definition 2.1.11 (maximal tuples)**.** For a relation $R :: T^2$ and a set $Y :: T$ we define

$$R \rhd Y =_{df} Y - |R\rangle Y \ ,$$

where "$-$" is set difference.

These are the Y-tuples that do not have an R-successor in Y, i.e., are not dominated by any tuple in Y. The mnemonic behind the $\rhd$ symbol is that in an order diagram for a preference relation R the maximal tuples within Y are the peaks in Y; rotating the diagram clockwise by 90° puts the peaks to the right. Hence $R \rhd Y$ might also be read as "R-peaks in Y".

2.2 Preference Relations and Constructors

Building on the theoretical foundations of typed binary relations we will now consider our concrete application, the database preferences. Even though we have introduced operations like the maximum operator in a point-free algebraic manner, we will now take one step back and use the point-wise definitions of preferences from the application related contexts. There, typically a more explicit way of notation is used, which corresponds to point-wise and quantified definitions.

To bridge the gap to our point-free relational approach we will show simpler definitions of some of these operations using the join operator. This simplification sketches the idea of the more general point-free and algebraic approach to the preference framework, which will be presented in the next chapter.

To formally connect our calculus to the applications, we will adapt our notation and terminology to the foundational paper on database preferences [Kie02]. To take account of the recent developments in this area, we will introduce the base preference constructors as they are used in the commercial implementation *Exasolution Skyline* [MKEK15] and the freely available *rPref* package [Roo15f].

Primarily we introduce the general notion of a preference in a notational fashion closely adapted to the [Kie02] and *SV semantics* introduced in [Kie05]. We use the typing mechanism presented in the previous section to formally connect a preference relation with its domain.

Definition 2.2.1 (preference)**.** A preference $<_P :: T^2$ is a strict partial order (irreflexive and transitive) defined on the domain values D_T of the attributes contained in T. The predicate $x <_P y$ is interpreted as "I like y more than x".

Every preference is associated with an SV relation $\cong_P$. This has to be an equivalence relation on the domain D_T. The equivalence classes contain "equally good" tuples (SV is short for "substitutable values"). It must fulfil the compatibility conditions

$$\begin{aligned} \forall x, z :: T : \exists y :: T : \; & (\, (x <_P y \wedge y \cong_P z) \;\Rightarrow\; x <_P z \,) \;\wedge \\ & (\, (x \cong_P y \wedge y <_P z) \;\Rightarrow\; x <_P z \,) \; , \\ \forall x, y :: T : \; & x <_P y \;\Rightarrow\; \neg(x \cong_P y) \; . \end{aligned}$$

The intuition of SV relations is, that in a predicate $x <_P z$ the tuple x can be substituted by y if y is SV-equivalent to x. By the compatibility relations it is guaranteed that the truth value of this predicate is invariant under such a substitution.

2.2.1 Base Preferences

Base preferences are the fundamental building blocks to construct preference orders. They are used to model a user wish for low or high values in numerical domains. Additionally, user wishes for tuples fulfilling a given Boolean predicate can be modelled within a preference.

Definition 2.2.2 (low and high)**.** The numerical low and high constructors define the wish for maximal or minimal values of this domain. Let $\phi : D_T \to \mathbb{R}$ be a function mapping from a type domain D_T to a numerical domain. We define the following preferences for all $x, y :: T$:

$$\begin{aligned} x <_{\mathsf{low}(\phi)} y &\iff \phi(y) < \phi(x) \; , \\ x <_{\mathsf{high}(\phi)} y &\iff \phi(y) > \phi(x) \; . \end{aligned}$$

The corresponding SV relation for $P = \mathsf{low}(\phi)$ and $P = \mathsf{high}(\phi)$ is given by

$$x \cong_P y \iff \phi(x) = \phi(y) \; .$$

To formally specify ϕ in the above definition we will use attributes (i.e., singleton types), or arithmetic/logical expressions of attributes. These can directly occur in the base preference constructors low and high. With $\mathsf{low}(\mathit{expr}) :: T^2$ where *expr* is an expression (e.g., $\mathit{expr} = A \cdot B$) over attributes in T we represent a preference $\mathsf{low}(\phi)$ where ϕ is defined as the evaluation function of the arithmetical/logical term *expr* over the given data set $M :: T$ within a maximum expression $\mathsf{low}(\mathit{expr}) \rhd M$.

The same convention applies to the implementations of preferences in *rPref* and *Exasolution Skyline* and hence this notation establishes a tight connection between the formal notation and the syntax of existing preference query languages. The definitions of low and high have corresponding keywords (or functions) in Exasolution Skyline and rPref. The ϕ function may be an arbitrary numerical SQL expression or an R expression, respectively. For example, `low(2*A+B)` constructs a low preference in both of these languages, where the optimality of a tuple t is induced by the minimality of $2 \cdot \pi_A(t) + \pi_B(t)$.

These constructors generalize the "highest" and "lowest" constructors from [Kie02]. In this definition, lowest(A) constructs a preference on a single attribute A, i.e., A is a singleton type and $\phi : D_A \to \mathbb{R}$ is defined as the identity $\phi(x) = x$. This implies that the type domain D_A has to be numerical.

We give a first example of a low preference on a numerical domain.

Example 2.2.3. Let $T = A \bowtie B$ be a type with the domain $D_T = \mathbb{N} \times \mathbb{N}$. Assume a data set $M :: T$ with $M = \{(0,0), (0,1), (1,1)\}$ where the first component of every tuple

corresponds to the attribute A and the second to B. For a preference $<_P$ with $P = \mathsf{low}(A)$ we obtain the maxima in M with

$$<_P \rhd M = M - |<_P\rangle M = M - \{(1,1)\} = \{(0,0),(0,1)\} \ .$$

To obtain a preference from a logical expression we introduce a constructor for Boolean preferences in the following definition.

Definition 2.2.4 (Boolean preference)**.** The is_true constructor defines the wish for tuples where a given logical predicate evaluates to true. Let $\rho : D_T \to \{\mathsf{false}, \mathsf{true}\}$ be a mapping from a type domain D_T to the logical domain. We define the following preference for all tuples $x, y :: T$ by

$$x <_{\mathsf{is\_true}(\rho)} y \iff \rho(x) = \mathsf{false} \wedge \rho(y) = \mathsf{true} \ .$$

The corresponding SV relation is given by

$$x \cong_{\mathsf{is\_true}(\rho)} y \iff \rho(x) = \rho(y) \ .$$

In the Exasolution Skyline implementation such a preference is constructed by just typing an arbitrary SQL expression which results in a logical value. In rPref the constructor for Boolean preferences is `true(expression)` where `expression` corresponds to ρ. Note that in the R language `true` is not a reserved word, instead only the word `TRUE` (in capitals) represents the logical value true.

These Boolean preferences generalise the "pos" constructor in [Kie02], which is restricted to logical conditions expressing "is contained in". Formally, a preference $\mathrm{pos}(A, M)$ is defined for a single attribute A and a set $M :: A$. We have $\mathrm{pos}(A, M) = \mathsf{is\_true}(\rho_M)$ where

$$\rho_M(x) = \begin{cases} \mathsf{true} & \text{if } x \in M \ , \\ \mathsf{false} & \text{otherwise} \ . \end{cases}$$

Our notion of the *logical domain* always refers to the set $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$. In contrast to this, databases usually support a trivalent logic with a third logical value "unknown". This value is returned for all predicates where missing values (`NULL` in SQL or `NA` in R) occur. All our calculi can be extended to this trivalent logic. In consequence, the dominance criterion $x <_P y$, where on of the tuples $\{x, y\}$ contains some missing values, has the logical value "unknown". This implies that x and y are incomparable w.r.t. $<_P$. Hence such tuples with missing values cannot be dominated and they will always occur in the corresponding maximal set.

While the trivalent logic is theoretically sound and well studied, in practice it is often not that what the user actually needs. One reason is, that `NULL` is commonly misused as an "empty instance" instead of a missing value. As a remedy, in [ERW+12] a formalism for a more sophisticated handling of NULL-values is suggested. This approach pragmatically solves the problem for certain applications, but in generally it is not consistent to the NULL-handling in established database theory. Hence it is neither implemented in Exasolution Skyline nor in rPref and we will not go into the details of this.

2.2.2 Complex Preferences

The orderings induced by the preference constructors from the section above are isomorphic to linear orderings. Hence such a preference query has the same expressiveness as the `ORDER BY` clause of standard SQL. To see the power of the preference concept, we will consider Pareto preferences subsequently. Regarding the terminology and symbols we will follow the concept of complex preferences as introduced in [Kie02].

Definition 2.2.5 (Pareto preference)**.** Given two preferences P and Q with $<_P, \cong_P :: T_1^2$ and $<_Q, \cong_Q :: T_2^2$, the Pareto operator $\otimes$ is defined as follows. For all $x, y :: T_1 \bowtie T_2$ we define, where $x_i = \pi_{T_i}(x)$ and $y_i = \pi_{T_i}(y)$ for $i = 1, 2$:

$$\begin{aligned} x <_{P\otimes Q} y \;\; \Leftrightarrow_{df} \; & (x_1 <_P y_1 \wedge (x_2 \cong_Q y_2 \vee x_2 <_Q y_2)) \vee \\ & (x_2 <_Q y_2 \wedge (x_1 \cong_P y_1 \vee x_1 <_P y_1)) \end{aligned}$$

The associated SV relation is inherited from the SV relations of the operands:

$$x \cong_{P\otimes Q} y \;\; \Leftrightarrow_{df} \;\; x_1 \cong_P y_1 \wedge x_2 \cong_Q y_2 \; .$$

For the resulting type we have $<_{P\otimes Q}, \cong_{P\otimes Q} :: T_1 \bowtie T_2$.

This means that $y = y_1 \bowtie y_2$ dominates $x = x_1 \bowtie x_2$, if and only if x is better than or equal to y (in the sense of SV-equivalence) w.r.t. both preferences $<_P$ and $<_Q$ and strictly better w.r.t. at least one of them. If both preferences are either low, high preferences or (nested) Pareto compositions of these base preferences, this concept is identical to Skylines as defined in [BKS01]. As the Pareto preference constructor also supports other arguments, e.g., Prioritization preferences or Boolean preferences, the Pareto preference is a generalization of the Skyline operator introduced in the pioneering paper [BKS01].

We want to transform the above definition into a more concise and readable variant using the $\bowtie$ operator. Therefore we calculate, where $\bowtie$ binds tighter than $\cup$,

$$\begin{aligned} & x <_{P\otimes Q} y \\ \Leftrightarrow \quad & \{\![\text{ definition }]\!\} \\ & (x_1 <_P y_1 \wedge (x_2 \cong_Q y_2 \vee x_2 <_Q y_2)) \vee (x_2 <_Q y_2 \wedge (x_1 \cong_P y_1 \vee x_1 <_P y_1)) \\ \Leftrightarrow \quad & \{\![\text{ definition of relational union }]\!\} \\ & (x_1 <_P y_1 \wedge (x_2\,(\cong_Q \cup <_Q)\,y_2)) \vee (x_2 <_Q y_2 \wedge (x_1(\cong_P \cup <_P)\,y_1)) \\ \Leftrightarrow \quad & \{\![\; x_i = \pi_{T_i}(x),\; y_i = \pi_{T_i}(y) \text{ and definition of join }]\!\} \\ & x\,(<_P \bowtie (\cong_Q \cup <_Q))\,y \vee x\,((\cong_P \cup <_P) \bowtie <_Q)\,y \\ \Leftrightarrow \quad & \{\![\text{ definition of relational union }]\!\} \\ & x\,(<_P \bowtie (\cong_Q \cup <_Q) \;\cup\; (\cong_P \cup <_P) \bowtie <_Q)\,y \; . \end{aligned}$$

This yields a point-free definition of the Pareto operation

$$<_{P\otimes Q} = (<_P \bowtie (\cong_Q \cup <_Q) \;\cup\; (\cong_P \cup <_P) \bowtie <_Q) \; .$$

For the SV relation we just get the relational intersection, i.e., we formally have $\cong_{P\otimes Q} = \cong_P \cap \cong_Q$.

In the applications, the Pareto operator is used for *equal important* user wishes. A typical use case is given in the following example.

Example 2.2.6. Consider the R data set `mtcars`. The goals of high horse power and low fuel consumption (corresponding to a high "miles per gallon" value) tend to be conflicting, as the points in Figure 2.1 are placed roughly around a line from top left to bottom right. The highlighted points are the Skyline. Formally this set corresponds to

$$<_{\mathsf{high}(hp)\otimes\mathsf{high}(mpg)} \rhd \text{mtcars} \; .$$

All cars in the Skyline are not dominated by any other car in the data set. Visually this means that the top right area of every Skyline point is empty. All points which are not in the Skyline are dominated by some point which is top right from them, cf. Figure 2.1.

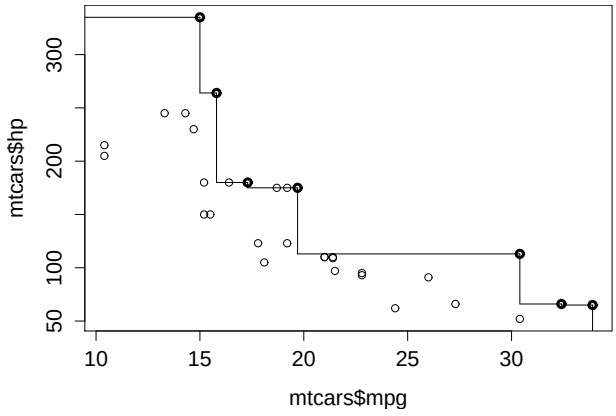


Figure 2.1: The dimensions *horsepower* and *miles per gallon* (inverse fuel consumption) of the R data set `mtcars`. The Skyline showing the Pareto optima w.r.t. both dimensions (Pareto frontier) is highlighted.

The Pareto operator has a number of pleasant properties, e.g., it is commutative, associative and preserves preferences. Outgoing from the point-free formulation which we derived above, they follow straight forward from the properties of the join, union, etc.

Another important complex preference constructor is the prioritisation preference, which is equivalent to the lexicographic order.

Definition 2.2.7 (prioritisation preference)**.** Given two preferences P and Q with $<_P, \cong_P :: T_1^2$ and $<_Q, \cong_Q :: T_2^2$, the *Prioritisation* operator & is defined as follows. For all $x, y :: T_1 \bowtie T_2$ we define, where $x_i = \pi_{T_i}(x)$ and $y_i = \pi_{T_i}(y)$ for $i = 1, 2$:

$$x <_{P\&Q} y \;\Leftrightarrow_{df}\; x_1 <_P y_1 \vee (x_1 \cong_P y_1 \wedge x_2 <_Q y_2) \;,$$
$$x \cong_{P\&Q} y \;\Leftrightarrow_{df}\; x_1 \cong_P y_1 \wedge x_2 \cong_Q y_2 \;.$$

Again we implicitly derive a point-free variant of the prioritization operator by calculating

$$\begin{aligned}
& x <_{P\&Q} y \\
\Leftrightarrow\quad & \{\![\text{ definition }]\!\} \\
& x_1 <_P y_1 \vee (x_1 \cong_P y_1 \wedge x_2 <_Q y_2) \\
\Leftrightarrow\quad & \{\![\text{ definition of universal relation }]\!\} \\
& (x_1 <_P y_1 \wedge x_2 \top_{T_1} y_1) \vee (x_1 \cong_P y_1 \wedge x_2 <_Q y_2) \\
\Leftrightarrow\quad & \{\![\; x_i = \pi_{T_i}(x),\; y_i = \pi_{T_i}(y) \text{ and definition of join }]\!\} \\
& (x\,(<_P \bowtie \top_{T_2})\,y) \wedge (x\,(\cong_P \bowtie <_Q)\,y) \\
\Leftrightarrow\quad & \{\![\text{ definition of relational union }]\!\} \\
& x\,(<_P \bowtie \top_{T_2} \cup \cong_P \bowtie <_Q)\,y \;.
\end{aligned}$$

The prioritisation also is commutative, associative and preserves preferences. In addition to this, as we will see later, also weak strict orders are preserved. The prioritisation is used to compose preferences of decreasing importance, i.e., in $<_{P\&Q}$ the preference $<_P$ is more important than $<_Q$.

Another complex preference, which is a unary operator, is the inverse preference.

Definition 2.2.8 (inverse preference)**.** For a preference $<_P :: T^2$ the inverse preference is given by the inverse relation. Formally we define $<_{P^{-1}} =_{df} (<_P)^{-1}$ and $\cong_{P^{-1}} =_{df} \cong_P$.

Less frequently, preferences are composed using *intersection* and *union*. Their point-free definitions are simply given by $<_{P \blacklozenge Q} =_{df} <_P \cap <_Q$ for the intersection and $<_{P+Q} =_{df} <_P \cup <_Q$ for the union. In both cases the associated SV relation is the relational intersection $\cong_P \cap \cong_Q$, equivalently to those of Pareto composition and Prioritisation. Note that the intersection preserves preferences while the union does not in general. For example, for any preference $<_P$ with $<_P \neq \emptyset$ the relation $<_{P+P^{-1}}$ is not transitive.

Because Pareto composition and prioritisation have these pleasant properties and the intuitive interpretation of composing "equally important" and "less important" wishes, they are the most important binary preference operators. Together with the inverse preference and the base preferences from the previous section they form the preference query language which is supported by Exasolution Skyline. In this language the prioritisation is realized with the keyword `prior to`, the Pareto composition by using `plus` and the inverse preference is denoted by a preceding keyword `reverse`. For example the preference $<_P$ with

$$P = \mathsf{low}(a) \otimes (\mathsf{high}(b) \,\&\, \mathsf{is\_true}(c = 1))^{-1}$$

corresponds to the term `low(a) plus reverse(high(b) prior to c=1)`. Note that logical expressions like `c=1` are implicitly interpreted as Boolean preferences by Exasolution Skyline. Implicitly we assume that all attributes have numerical domains. For details regarding the preference query language of Exasolution Skyline, we refer to the documentation [Exa15].

In the rPref package [Roo15f] all mentioned preference operators are supported. The operators $\otimes$ and & are denoted by `*` and `&` and the inverse preference with a preceding `-` operator. The above example $<_P$ is expressed by the term `low(a) * -(high(b) & true(c==1))`. Note that the comparison operator is `==` in R. The operators $\blacklozenge$ and $\cup$ are also supported (via `|` and `+`), even though they are rarely used.

CHAPTER 3

An Algebraic Calculus

Algebraic structures like semirings and relation algebras are well-studied concepts for concise and elegant calculi in theoretical computer science. They are predestined to study which assumptions are really necessary for the proofs and they allow applying off-the-shelf theorem provers. On top of an abstract relation algebra we introduce the *join algebra*, which is the abstract counterpart of typed relations, and a join operator defined on them. We show how database preferences are expressed within the join algebra and derive some algebraic laws for them. Since we know from the previous chapter that the calculation of maximal elements can be expressed by a diamond operation (inverse image) of a data set, we will make use of the known theory around that.

3.1 Semirings, Abstract Relation Algebras and the Join Algebra

The join algebra was initially introduced in [MRE12, MR12b]. A comprehensive and extended definition was given in [MR15], from which we mainly adapt. Before introducing the join algebra we have to define the underlying structure of a semiring and an abstract relation algebra. We also have to translate the typing mechanism for tuples to the algebraic setting.

3.1.1 Semirings

At first we formally define the structure of an idempotent semiring, which turned out to be a very useful framework for many algebraic calculi in theoretical computer science.

Definition 3.1.1 ((idempotent) semiring)**.** A *semiring* is a quintuple $(S, +, 0, \cdot, 1)$, consisting of a set S of elements together with binary operations $+$ of *choice* and $\cdot$ of *composition*. Both are required to be associative, choice also to be commutative. Moreover, composition has to distribute over choice in both arguments. Finally, there have to be units 0 for choice and 1 for composition, where 0 is an annihilator w.r.t. composition. If $+$ is also idempotent, this structure is called an idempotent semiring.

For example $(\mathbb{N}_0, +, 0, \cdot, 1)$ and $(\mathbb{R}, +, 0, \cdot, 1)$ are semirings which are not idempotent. The paths over a set M form the idempotent semiring $(\mathcal{P}(M^+), \cup, \varnothing, ;, M)$, where $\cup$ models branching paths and $;$ is the concatenation of paths. The set of non-empty paths is

$M^+ = \bigcup_{i\geq 1} M^i$ and $\mathcal{P}(\cdot)$ is the power set operator. As another example, the set of languages over an alphabet Σ can be represented by an idempotent semiring via $(\mathcal{P}(\Sigma^*), \cup, \varnothing, ;, \{\epsilon\})$ where ϵ is the empty word and $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$. Another simple semiring is the Boolean algebra $(\mathbb{B}, \vee, \mathsf{false}, \wedge, \mathsf{true})$ with $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$.

In the following we will restrict our attention to binary homogeneous relations over a set. They form an idempotent semiring with choice $\cup$ and composition ;, which have the empty set and the identity relation as their respective units.

Definition 3.1.2 ((atomic) test, subsumption order)**.** Every idempotent semiring induces a *subsumption order* by $x \leq y \Leftrightarrow x+y = y$. A *test* is an element $x \leq 1$ that has a complement $\neg x$ relative to 1, i.e., that satisfies

$$x + \neg x = 1 \ , \quad x \cdot \neg x = 0 = \neg x \cdot x \ .$$

A test p is said to be *atomic*, if there are no elements x, y with $0 \neq x \neq y \neq 0$ such that $p = x + y$.

The subsumption order will turn out to be an important tool for algebraic proofs. We will often use antisymmetry, i.e., $x \leq y \wedge y \leq x \Leftrightarrow x = y$.

It is well known (e.g., [MB85]) that the complement is unique when it exists and that the set of all tests forms a Boolean algebra with + as join and $\cdot$ as meet. Tests are used to represent subsets or assertions in an algebraic way. In the semiring of binary relations over a set M the tests are subidentities, i.e., subsets of the identity relation, of the form $I_N =_{df} \{(t,t) \mid t \in N\}$ for some subset $N \subseteq M$ and hence in one-to-one correspondence with the subsets of M.

With regard to our application, we will use tests as an abstract representation of data sets or sets of tuples. Atomic tests are used to represent single tuples. Due to this abstraction, the difference between a tuple x and a set $\{x\}$, containing that single tuple, vanishes.

In the following we will use small letters a, b, c, d at the beginning of the alphabet to denote arbitrary semiring elements (mainly relations) and p, q, r, s to denote tests. The difference of two tests p, q can be defined as $p - q =_{df} p \cdot \neg q$. This corresponds to the usual set difference.

Complements and the subsumption ordering are connected by the *shunting rule*

$$p \cdot q \leq r \Leftrightarrow p \leq \neg q + r \ .$$

By setting $p = 1$ and applying shunting twice we can derive the *contraposition* rule

$$q \leq r \Leftrightarrow \neg r \leq \neg q \ .$$

The composition of tests and general elements is used for domain or range restrictions. For example, when a is a relation and p, q are tests, $p \cdot a \cdot q$ consists of the subrelations of a having initial points in the corresponding set of p and ending points in that of q.

Using these properties we will subsequently give an algebraic definition of the tests $\langle a|q$ and $|a\rangle q$ that represent the (inverse) image of the set represented by q w.r.t. a. Equivalently we can describe the inverse image $|a\rangle q$ as the set of initial points in $a \cdot q$ and the image $\langle a|q$ as the ending points in $q \cdot a$.

Definition 3.1.3 (forward/backward diamond)**.** Following [DMS06], the *forward diamond* is axiomatized by the universal property

$$|a\rangle q \leq p \Leftrightarrow a \cdot q \leq p \cdot a \cdot q \Leftrightarrow a \cdot q \leq p \cdot a \ .$$

For the *backward diamond* we symmetrically define

$$\langle a|q \leq p \Leftrightarrow q \cdot a \leq q \cdot a \cdot p \Leftrightarrow q \cdot a \leq a \cdot p \ .$$

The forward and backward diamonds fulfil the following useful algebraic properties:

$$\begin{aligned}
|a\rangle 0 &= 0 \ , & \langle a|0 &= 0 \ , \\
|a+b\rangle p &= |a\rangle p + |b\rangle p \ , & \langle a+b|p &= \langle a|p + \langle b|p \ , \\
|a\rangle(p+q) &= |a\rangle p + |a\rangle q \ , & \langle a|(p+q) &= \langle a|p + \langle a|q \ , \\
|r \cdot a\rangle p &= r \cdot |a\rangle p \ , & \langle a \cdot r|p &= r \cdot \langle a|p \ , \\
|a \cdot r\rangle p &= |a\rangle(r \cdot p) \ , & \langle r \cdot a|p &= \langle a|(r \cdot p) \ , \\
|a \cdot b\rangle p &= |a\rangle(|b\rangle p) \ , & \langle a \cdot b|p &= \langle a|(\langle b|p) \ .
\end{aligned}$$

The second and third line, showing distributivity in both arguments, also imply that diamond is isotone (i.e., monotonically increasing) in both arguments:

$$\begin{aligned}
a \leq b &\Rightarrow |a\rangle p \leq |b\rangle p \ , & a \leq b &\Rightarrow \langle a|p \leq \langle b|p \ , \\
p \leq q &\Rightarrow |a\rangle p \leq |a\rangle q \ . & p \leq q &\Rightarrow \langle a|p \leq \langle a|q \ .
\end{aligned}$$

Using the diamonds we can also express the starting points of a relation by the test $|a\rangle 1$ and the ending points by $\langle a|1$. They are often called *domain* and *codomain*. To avoid confusion with the domain of a type, we will not use these terms in this thesis.

3.1.2 Typed Elements

Analogously to the typing mechanism for sets and relations from Section 2.1 we introduce type assertions for tests and general elements. Additionally to their typing we extend this mechanism to *sub-types* which will be helpful to restrict algebraic operations to subsets of the domain, e.g., a given data set.

Let again be $\mathcal{A}$ the set of all attributes and $T \subseteq \mathcal{A}$ a type. With this we associate a test 1_T representing its domain D_T. The 1 without index represents the universe $\mathcal{U}$, i.e., we have $1 = \sum_{T \in \mathcal{A}} 1_T$. This 1 fulfils the properties of a neutral element w.r.t. $\cdot$ for elements of all types.

In the following we give an algebraic definition of type assertions for tests or general elements. General elements will be used for preference relations; in this case the greatest element $\top_T$ for a type T w.r.t. the subsumption order $\leq$ represents the universal relation $D_T \times D_T$. The element 0_T is the smallest element and represents the empty set $\varnothing$. Note that this is also typed.

Definition 3.1.4 ((sub)type assertions)**.** For a test p and a general element a we define the type assertions

$$p :: T \Leftrightarrow_{df} p \leq 1_T \ , \quad a :: T^2 \Leftrightarrow_{df} a \leq 1_T \cdot a \cdot 1_T \ .$$

For a test $r :: T$ we define the induced subtype $T[r]$ by the type assertions

$$p :: T[r] \Leftrightarrow_{df} p \leq r \ , \quad a :: T[r]^2 \Leftrightarrow_{df} a \leq r \cdot a \cdot r \ .$$

For subtypes, the special elements are given by

$$0_{T[r]} = 0_T \ , \quad 1_{T[r]} = r \ , \quad \top_{T[r]} = r \cdot \top_T \cdot r \ .$$

For a element x which is either a test or a general element of a (sub)type T we use the notation

$$x :: T^{(2)} \Leftrightarrow_{df} x :: T \vee x :: T^2 \ .$$

Note that every test is a special case of a general element, i.e., formally it holds that $x :: T^{(2)} \Leftrightarrow x :: T^2$. In contrast to this fact, we will use $x :: T^2$ only if x is intended to represent a relation. Typically an element $a :: T^2$ is either empty ($a = 0_T$) or not a sub-identity ($a \sqcap 1_T \neq 0$). Especially in our domain of preferences we deal with irreflexive relations, i.e., every preference relation is either empty or not a test. Elements $x :: T^{(2)}$ are used to represent *either* sets of tuples *or* relations on tuples (and not mixtures of both).

The global identity 1 is not typeable as long as types are restricted to subsets $T \subseteq \mathcal{A}$. In [Roo12] we extend the typing mechanism to *multitypes* where the power set $\mathcal{P}(\mathcal{A})$ becomes the type of 1. Such theoretical extensions of the calculus are not relevant for the following theorems and definitions. Hence we will not go into the details.

For sake of readability we introduce some notational conventions, abbreviating the typing of elements. For $x :: T^{(2)}$ we define

$$0_x =_{df} 0_T \ , \quad 1_x =_{df} 1_T \ , \quad \top_x =_{df} \top_T \ .$$

The definition of the identity on subtypes, i.e., $1_{T[r]} = r$ also implies that all complements are relative to r. For a test $p :: T[r]$ we have $\neg p = 1_{T[r]} \cdot \neg p = r - p$ just by neutrality of the identity w.r.t. $\cdot$.

With regard to the application to databases we use 1_T to represent the entire domain, e.g., $\mathbb{R}$ for the numerical data type `float`. Such a domain is in generally infinite. In contrast to this, a data set $r \leq 1_T$ is finite. Hence the associated sub-type domain $1_{T[r]}$ contains only finite elements, which makes many calculations easier.

3.1.3 Abstract Relation Algebra

In the explanation of our calculus above we already sketched the idea of our application. The abstract semiring elements are intended to represent data sets and preference relations in an abstract way. But up to now this just reflects our idea how to use it; formally we have not specified what $a + b$ or $a \cdot b$ actually is. For relations a, b these operations are intended to model the relational union and composition. In contrast to the previous chapter we do not want to define them explicitly, as this would require quantifiers. Instead of this we specify some axioms which ensure that these operations coincide with the usual union and composition. These axioms can be given in a quantifier-free (or point-free) way. This leads to the concept of an *abstract relation algebra*.

With this concept we embed the binary homogeneous relations from our application into the abstract setting. We preserve the point-free formalism throughout and obtain short axiomatizations which are well suited for automated theorem provers. We define the abstract relation algebra following [Mad97]:

Definition 3.1.5 (abstract relation algebra)**.** An *abstract relation algebra* is an idempotent semiring with additional operators $(...)^{-1}, \overline{(...)}$ for converse and complement, axiomatized by the Schröder equivalences and Huntington's axiom:

$$x \cdot y \leq z \Leftrightarrow x^{-1} \cdot \overline{z} \leq \overline{y} \Leftrightarrow \overline{z} \cdot y^{-1} \leq \overline{x} \ , \qquad x = \overline{\overline{x} + y} + \overline{\overline{x} + \overline{y}} \ .$$

For our applications, we additionally stipulate the Tarski rule

$$a \neq 0_a \Rightarrow \top_a \cdot a \cdot \top_a = \top_a \ ,$$

where $\top_a = \overline{0_a}$. □

We assume that our underlying semiring is an abstract relation algebra and each type domain is closed under converse and complement, i.e., for $x :: T^{(2)}$ we have also $x^{-1}, \overline{x} :: T^{(2)}$. Note that this also holds for sub-types $T[r]$, where the complement is relative to r.

For an easier notation, we introduce the meet operation and the difference between two elements as follows:

$$x \sqcap y =_{df} \overline{\overline{x} + \overline{y}} \ , \quad x - y =_{df} x \sqcap \overline{y} \ .$$

In case that x, y represent relations, these operations correspond to the relational intersection and difference. For tests $p, q \leq 1$ they coincide with composition and relative complement:

$$p \sqcap q = p \cdot q \ , \quad p - q = p \cdot \neg q \ .$$

For $a :: T^2$ the powers a^k (i.e., iterated composition) for $k \in \mathbb{N}_0$ are defined by

$$a^k =_{df} a^{k-1} \cdot a \ \text{ for } \ k \geq 1 \qquad \text{and} \qquad a^0 =_{df} 1_a \ .$$

3.1.4 Join Algebra

Up to now we introduced well-known structures. In the following we define the *join algebra*, extending the abstract relation algebra by an axiomatization of the join operation using the typing mechanism axiomatized above. This concept was originally introduced in [MRE12] and we follow the extended definition of [MR15].

Definition 3.1.6 (join algebra)**.** A *join algebra* is an abstract relation algebra with an additional binary operator $\bowtie$ satisfying the following requirements.

1. Join is associative, commutative and idempotent and distributes over choice + in both arguments. Hence $\bowtie$ is isotone in both arguments.
2. Join is zero-strict, i.e., we have $a \bowtie 0_{T'} = 0_{T \bowtie T'}$ for all $a :: T^{(2)}$.
3. If $a_i :: T_i^{(2)}$ with $i = 1, 2$ then $a_1 \bowtie a_2 :: (T_1 \bowtie T_2)^{(2)}$.
4. For types T_i with $i = 1, 2$ we have
$$1_{T_1 \bowtie T_2} = 1_{T_1} \bowtie 1_{T_2} \quad \text{and} \quad \top_{T_1 \bowtie T_2} = \top_{T_1} \bowtie \top_{T_2} \ .$$
5. Join and composition satisfy, for $a_i, b_i :: T_i^{(2)}$ with $i = 1, 2$ and disjoint T_i, the exchange law
$$(a_1 \bowtie a_2) \cdot (b_1 \bowtie b_2) = (a_1 \cdot b_1) \bowtie (a_2 \cdot b_2) \ .$$
6. The diamond operator respects joins of elements with disjoint types: for $a :: T_1^2, p :: T_1$ and $b :: T_2^2, q :: T_2$ with $T_1 \sqcap T_2 = \varnothing$ we have the exchange law
$$|a \bowtie b\rangle (p \bowtie q) \ = \ |a\rangle p \bowtie |b\rangle q \ .$$
7. If $a_1, a_2 :: T^{(2)}$, then $a_1 \bowtie a_2 = a_1 \sqcap a_2$.
8. Join and meet satisfy, for $a_i, b_i :: T_i^{(2)}$ with $i = 1, 2$ and disjoint T_i, the exchange law
$$(a \bowtie b) \sqcap (c \bowtie d) = (a \sqcap c) \bowtie (b \sqcap d) \ .$$

The typed relations from Section 2.1.3 fulfil all axioms of a join algebra. Therewith we conclude the definition of the algebraic foundations for our calculus. Subsequently, we give a first property for the complements of joined sets and relations.

Lemma 3.1.7. *Let $a \bowtie b :: T_a^2 \bowtie T_b^2$ and $p \bowtie q :: T_a \bowtie T_b$ with $T_a \sqcap T_b = \varnothing$. For the complements of tests and the general complements we have, where $\neg$ binds tighter than $\bowtie$ and $\bowtie$ tighter than $+$,*

$$\overline{a \bowtie b} = \overline{a} \bowtie \top_b \;+\; \top_a \bowtie \overline{b} \;,$$

$$\neg(p \bowtie q) = \neg p \bowtie 1_b \;+\; 1_a \bowtie \neg q \;.$$

Proof. For the first claim we have to show that the following holds:

$$(a \bowtie b) \sqcap \overline{a \bowtie b} = 0_a \bowtie 0_b \quad \wedge \quad a \bowtie b \;+\; \overline{a \bowtie b} = \top_a \bowtie \top_b \;.$$

For the first equation we calculate

$$\begin{aligned}
& (a \bowtie b) \sqcap (\overline{a} \bowtie \top_b \;+\; \top_a \bowtie \overline{b}) \\
= \quad & \{\![\sqcap \text{ distributes over } +, \text{ exchange law for } \sqcap \text{ and } \bowtie]\!\} \\
& (a \sqcap \overline{a}) \bowtie (b \sqcap \top_b) \;+\; (a \sqcap \top_a) \bowtie (b \sqcap \overline{b}) \\
= \quad & \{\![\text{ properties of intersection and complements }]\!\} \\
& 0_a \bowtie b \;+\; a \bowtie 0_b \\
= \quad & \{\![\text{ 0-strictness of } \bowtie]\!\} \\
& 0_a \bowtie 0_b \;.
\end{aligned}$$

For the second equation we have

$$\begin{aligned}
& a \bowtie b \;+\; \overline{a} \bowtie \top_b \;+\; \top_a \bowtie \overline{b} \\
= \quad & \{\![\; a \bowtie \overline{b} \leq \top_a \bowtie \overline{b} \text{ by subsumption order and } \top_a \text{ is greatest element }]\!\} \\
& a \bowtie b \;+\; a \bowtie \overline{b} \;+\; \overline{a} \bowtie \top_b \;+\; \top_a \bowtie \overline{b} \\
= \quad & \{\![\text{ distributivity of } \bowtie \text{ over } +]\!\} \\
& a \bowtie (b + \overline{b}) \;+\; \overline{a} \bowtie \top_b \;+\; \top_a \bowtie \overline{b} \\
= \quad & \{\![\text{ properties of complement }]\!\} \\
& a \bowtie \top_b \;+\; \overline{a} \bowtie \top_b \;+\; \top_a \bowtie \overline{b} \\
= \quad & \{\![\text{ distributivity of } \bowtie \text{ over } +, \text{ properties of complement }]\!\} \\
& \top_a \bowtie \top_b \;+\; \top_a \bowtie \overline{b} \\
= \quad & \{\![\text{ subsumption order and } \top_b \text{ is greatest element }]\!\} \\
& \top_a \bowtie \top_b \;.
\end{aligned}$$

This shows the first claim. The second one is analogous with 1_x instead of $\top_x$ and $\neg(\cdot)$ instead of $\overline{(\cdot)}$. □

In the following we proceed with our application of database preferences.

3.2 Algebraic Representation of Preferences

In Section 2.2 we already introduced preferences in the notational fashion of [Kie02]. In the following we will adapt this to the algebraic calculus introduced in this chapter.

With the derivation of point-free representations of complex preferences in Section 2.2.2 we already set the direction to a more concise formalism where the quantifiers just occur implicitly in the basic definitions of the relational calculus. With the following axiomatizations we generally avoid explicit definitions and reason about preferences at a more abstract level.

3.2.1 Basic Definitions and Properties

We give an algebraic redefinition of a preference replacing Definition 2.2.1. Moreover we introduce *layered preferences*, which are mathematically strict weak orders and thus a subclass of preferences.

Definition 3.2.1 ((layered) preference)**.** A relation $a :: T^2$ is a *preference* if and only if it is irreflexive and transitive, i.e., $a \sqcap 1_T = 0$ and $a^2 \leq a$.

Every preference a will be associated with an SV relation s_a. This has to be an equivalence relation on the domain of a, i.e., we require $1_a \leq \mathsf{s}_a$, $\mathsf{s}_a^2 \leq \mathsf{s}_a$ and $\mathsf{s}_a = \mathsf{s}_a^{-1}$. The equivalence classes contain "equally good" objects (SV is short for "substitutable values"). It must fulfil the compatibility conditions

$$\mathsf{s}_a \cdot a \leq a \ , \quad a \cdot \mathsf{s}_a \leq a \ .$$

A preference a is the special case of a *layered preference*, if and only if additionally *negative transitivity* $(\overline{a})^2 \leq \overline{a}$ holds. If the SV relation is not stated explicitly, then it is defined to be the identity, i.e., we set $\mathsf{s}_a = 1_a$. □

Compared to Definition 2.2.1, it becomes visible how this notation simplifies the formulation of the compatibility conditions. The third compatibility condition of Definition 2.2.1 translates to $a \sqcap \mathsf{s}_a = 0_a$ but we can omit this condition, as it is a consequence of the definition. We show this in the following Lemma from [MR15].

Lemma 3.2.2. *Compatibility of* $a :: T_a^2$ *and* $\mathsf{s}_a :: T_a^2$ *implies* $\mathsf{s}_a \sqcap a = 0_a$.

Proof. We use the Dedekind rule

$$c \cdot d \sqcap e \leq (c \sqcap e \cdot d^{-1}) \cdot d \ ,$$

which follows from the Schröder rule and Huntington's Axiom, cf. [SS93]. We calculate

$$\begin{array}{ll} & \mathsf{s}_a \sqcap a \\ = & \quad \{\![\text{ neutrality of } 1 \ ]\!\} \\ & 1_a \cdot \mathsf{s}_a \sqcap a \\ \leq & \quad \{\![\text{ Dedekind rule }]\!\} \\ & (1_a \sqcap a \cdot \mathsf{s}_a^{-1}) \cdot \mathsf{s}_a \\ = & \quad \{\![\text{ symmetry of } \mathsf{s}_a \ ]\!\} \\ & (1_a \sqcap a \cdot \mathsf{s}_a) \cdot \mathsf{s}_a \\ \leq & \quad \{\![\text{ compatibility of } a \text{ with } \mathsf{s}_a \ ]\!\} \\ & (1_a \sqcap a) \cdot \mathsf{s}_a \\ = & \quad \{\![\text{ by irreflexivity of } a \text{ we have } 1_a \sqcap a = 0_a \ ]\!\} \\ & 0_a \ . \end{array}$$

□

Not all relational predicates have point-free representations. Especially for the examples, but also for some theorems, we will need tuple-wise comparisons. Analogous to the relational notation we use for tuples (atomic tests) $u, v :: T$ and a preference $a :: T^2$ the predicate

$$u \, a \, v \ \Leftrightarrow_{df} \ u \cdot a \cdot v \neq 0$$

which expresses that u is a-related to v, i.e., v is better than u w.r.t. a. To express that tuples are not a-related we use $\neg(u \, a \, v)$ or, equivalently, $(u \, \overline{a} \, v)$. By convention, we will use small letters u, v, w, x, y, z for tuples in the following.

In the preference literature, layered preferences are also called *weak order preferences (WOP)*, cf. [Kie02]. The term *layered* preference is motivated by the "layered structure" which is induced by such relations. This is precisely stated in the following corollary where $\mathbb{N}$ are the natural numbers without 0.

Corollary 3.2.3. *Assume a countable domain D_T. A relation $a :: T^2$ is a layered preference if and only if there exists a measure function $f_a : 1_T \to \mathbb{N}$ such that*

$$u\, a\, v \Leftrightarrow f_a(u) < f_a(v) \ .$$

This corollary is proved in [Fis70], Theorem 2.2. The preimages $f_a^{-1}(k)$ with ascending k for $k = 1, 2, ...$ form a-related layers, i.e., we have

$$\forall k, l \in \mathbb{N} \text{ with } k < l : \forall u, v : |a\rangle(f_a^{-1}(l)) = f_a^{-1}(k) \ .$$

If we use the induced equivalence relation of f_a as SV relation s_a we have $(u\, \mathsf{s}_a\, v) \Leftrightarrow f_a(u) = f_a(v)$. Algebraically this means that $\mathsf{s}_a = \overline{a + a^{-1}}$ holds. We show in the following lemma from [MR15] that such an s_a fulfils indeed the compatibility conditions. Moreover, if such an s_a fulfils the property of an SV relation, we can infer the layered preference property.

Lemma 3.2.4. *Let $a :: T^2$ be a preference. Then we have:*

$$a \text{ is a layered preference } \iff \mathsf{s}_a = \overline{a + a^{-1}} \text{ is a valid SV relation for } a \ .$$

Proof. First, it is clear that s_a is an equivalence relation. We have to show that the property of a being layered, i.e., negative transitivity, is equivalent to the compatibility conditions of s_a with a. Formally the claim is

$$(\overline{a})^2 \le \overline{a} \iff \mathsf{s}_a \cdot a \le a \ \wedge \ a \cdot \mathsf{s}_a \le a \ .$$

Subsequently, we show both implications separately.

"⇒" By definition of s_a, the exchange law for complement and converse, and finally the infimum property we infer for $\mathsf{s}_a \cdot a$:

$$\mathsf{s}_a \cdot a = \left(\overline{a} \sqcap \overline{a^{-1}}\right) \cdot a = \left(\overline{a} \sqcap (\overline{a})^{-1}\right) \cdot a \le (\overline{a})^{-1} \cdot a \ .$$

We still have to show that $(\overline{a})^{-1} \cdot a \le a$. By the Schröder equivalences, this is equivalent to $\overline{a} \cdot \overline{a} \le \overline{a}$, which is just negative transitivity of a. Thus we conclude $\mathsf{s}_a \cdot a \le a$. For $a \cdot \mathsf{s}_a \le a$ an analogous argument holds; hence s_a is compatible with a.

"⇐" First we calculate, using the assumption $\mathsf{s}_a = \overline{a + a^{-1}}$ together with $a \sqcap \mathsf{s}_a = 0_a$ (Lemma 3.2.2) and $a \sqcap a^{-1} = 0_a$ (asymmetry of a, as a is a strict order):

$$\top_a = (a + a^{-1}) + \overline{a + a^{-1}} = a + a^{-1} + \mathsf{s}_a \ \Rightarrow \ a = \overline{a^{-1} + \mathsf{s}_a} \tag{3.1}$$

We have to show negative transitivity $(\overline{a})^2 \le \overline{a}$. We calculate:

$$\begin{aligned}
& (\overline{a})^2 \\
= \quad & \{\![\text{ Equation (3.1) }]\!\} \\
& (a^{-1} + \mathsf{s}_a)^2 \\
= \quad & \{\![\text{ distributivity }]\!\} \\
& (a^{-1})^2 + a^{-1} \cdot \mathsf{s}_a + \mathsf{s}_a \cdot a^{-1} + \mathsf{s}_a^2
\end{aligned}$$

$$
\begin{aligned}
&\leq \quad \{\![\text{ transitivity of } a^{-1}, \mathsf{s}_a \text{ and symmetry of } \mathsf{s}_a \text{ (i.e., } \mathsf{s}_a = \mathsf{s}_a^{-1}) \ ]\!\} \\
&\quad a^{-1} + (\mathsf{s}_a \cdot a)^{-1} + (a \cdot \mathsf{s}_a)^{-1} + \mathsf{s}_a \\
&\leq \quad \{\![\text{ compatibility conditions for } a \text{ and } \mathsf{s}_a \ ]\!\} \\
&\quad a^{-1} + a^{-1} + a^{-1} + \mathsf{s}_a \\
&= \quad \{\![\text{ idempotency of } + \ ]\!\} \\
&\quad a^{-1} + \mathsf{s}_a \\
&= \quad \{\![\text{ Equation (3.1) }]\!\} \\
&\quad \overline{a} \ .
\end{aligned}
$$

This shows the claim. □

Independently of the property of being a layered preference, the relation $\overline{a + a^{-1}}$ is an upper limit for the SV relation of all preference. We state this in the following lemma.

Lemma 3.2.5. *Let $a :: T^2$ be a preference. Then we have $\mathsf{s}_a \leq \overline{a + a^{-1}}$.*

Proof. By Lemma 3.2.2 we have $a \sqcap \mathsf{s}_a = 0_a$. This implies $\mathsf{s}_a \leq \overline{a}$. Using symmetry of s_a we have

$$\mathsf{s}_a = \mathsf{s}_a \sqcap \mathsf{s}_a^{-1} \leq \overline{a} \sqcap (\overline{a})^{-1} = \overline{a} \sqcap \overline{a^{-1}} = \overline{a + a^{-1}} \ . \qquad \square$$

Usually we use the SV relation $\mathsf{s}_a = \overline{a + a^{-1}}$ for all layered preferences. A measure function f_a that both induces the better-than-relation and the SV relations mostly meets the user expectation according to our experiences. This SV relation also applies to the definition of the SV relations for the low, high and is_true base preferences in Section 2.2.1. We even have that $\mathsf{low}(f_a)$ is equivalent to a, where a has the SV relation $\mathsf{s}_a = \overline{a + a^{-1}}$.

Still, smaller SV relations are possible. The identity 1_a is the smallest possible SV relation for every preference, which follows from the requirement that s_a has to be an equivalence relation. Using Lemma 3.2.5, we see that for every preference a all relations s_a with

$$1_a \leq \mathsf{s}_a \leq \overline{a + a^{-1}} \qquad \wedge \qquad \mathsf{s}_a \text{ is compatible with } a$$

are valid SV relations, which are usually many. In [Kie05], $\mathsf{s}_a = 1_a$ is called *trivial SV-semantics* and $\mathsf{s}_a = \overline{a + a^{-1}}$ is called *regular SV-semantics*.

But note that the relation $\overline{a + a^{-1}}$ may be excluded by the compatibility relations if a is not a layered preference. We will see this in the following counterexample.

Example 3.2.6. Let $r = x_1 + x_2 + x_3 :: T$ and $(x_1\, a\, x_2)$ be the only better-than-relation in a preference $a :: T^2$. For the relation $b = \overline{a + a^{-1}}$ it holds that $(x_2\, b\, x_3)$. The compatibility condition for a states

$$x_1\, a\, x_2 \ \wedge \ x_2\, \mathsf{s}_a\, x_3 \ \Rightarrow \ x_1\, a\, x_3 \ .$$

The right hand side of the implication is false. By setting $\mathsf{s}_a = b$ the left hand side is true, hence this would invalidate the implication. Thus b is not a valid SV relation in this case. To see that a is indeed not a layered preference we convince ourselves that the negative transitivity is violated. We have

$$x_1\, \overline{a}\, x_3 \ \wedge \ x_3\, \overline{a}\, x_2 \quad (\text{implying } x_1\, (\overline{a})^2\, x_2) \qquad \text{but} \quad \neg(x_1\, \overline{a}\, x_2) \ ,$$

and thus $(\overline{a})^2 \leq \overline{a}$ is not fulfilled.

3.2.2 Complex Preferences

In an analogous manner as for the definition of preference relations we will redefine the complex preferences, i.e., Definitions 2.2.5 and 2.2.7. Additionally we introduce the left and right semi-Pareto preferences. They are typically not used in practice, but they are a helpful concept in the theory. With the introduction of *Semi Skylines* in [End11] these are extensively studied.

Definition 3.2.7 (prioritisation and Pareto composition)**.** Let $a :: T_a^2$ and $b :: T_b^2$ be preferences with associated SV relations $\mathsf{s}_a :: T_a^2$ and $\mathsf{s}_b :: T_b^2$. The *prioritisation with SV* is given by ($\bowtie$ binds tighter than +):

$$a \,\&\, b :: (T_a \bowtie T_b)^2 \ ,$$
$$a \,\&\, b =_{df} a \bowtie \top_b \ + \ \mathsf{s}_a \bowtie b \ ,$$

whereas the *Pareto compositions with SV* are defined as

$$a \lhd\!\otimes\, b, a \otimes\!\rhd\, b, a \otimes b :: (T_a \bowtie T_b)^2 \ ,$$
$$a \lhd\!\otimes\, b =_{df} a \bowtie (b + \mathsf{s}_b) \ ,$$
$$a \otimes\!\rhd\, b =_{df} (a + \mathsf{s}_a) \bowtie b \ ,$$
$$a \otimes b =_{df} a \lhd\!\otimes\, b \ + \ a \otimes\!\rhd\, b \ .$$

The operators $\lhd\!\otimes$ and $\otimes\!\rhd$ are called *left (or right, respectively) semi-Pareto* and bind tighter than +. For $a \star b$ with $\star \in \{\&, \lhd\!\otimes, \otimes\!\rhd, \otimes\}$ we say that $a \star b$ is *SV-preserving* if and only if $\mathsf{s}_{a\star b} = \mathsf{s}_a \bowtie \mathsf{s}_b$. If the SV relation is not specified otherwise, we assume that $\mathsf{s}_{a\star b}$ is SV-preserving. One may always define other SV relations for complex preferences, as long as they fulfil the compatibility conditions of Definition 3.2.1.

Corollary 3.2.8. *The above notions are well-defined, i.e.,* $\mathsf{s}_a \bowtie \mathsf{s}_b$ *is indeed a valid SV relation for* $a \star b$ *with* $\star \in \{\&, \lhd\!\otimes, \otimes\!\rhd, \otimes\}$*.*

Proof. We show $(a \,\&\, b) \cdot (\mathsf{s}_a \bowtie \mathsf{s}_b) \leq a \,\&\, b$, where $\cdot$ binds tighter than $\bowtie$, by calculating

$$\begin{aligned}
& (a \,\&\, b) \cdot (\mathsf{s}_a \bowtie \mathsf{s}_b) \\
= \quad & \{\![\text{ definition }]\!\} \\
& (a \bowtie \top_b \ + \ \mathsf{s}_a \bowtie b) \cdot (\mathsf{s}_a \bowtie \mathsf{s}_b) \\
= \quad & \{\![\text{ distributive and exchange law }]\!\} \\
& a \cdot \mathsf{s}_a \bowtie \top_b \cdot \mathsf{s}_b + \mathsf{s}_a \cdot \mathsf{s}_a \bowtie b \cdot \mathsf{s}_b \\
\leq \quad & \{\![\ \top_b \text{ is greatest element in } T_b \text{ and idempotency of } \mathsf{s}_a \leq 1_a \ ]\!\} \\
& a \cdot \mathsf{s}_a \bowtie \top_b + \mathsf{s}_a \bowtie b \cdot \mathsf{s}_b \\
\leq \quad & \{\![\text{ compatibility conditions for } a \text{ and } b \ ]\!\} \\
& a \bowtie \top_b + \mathsf{s}_a \bowtie b = a \,\&\, b \ .
\end{aligned}$$

The other predicates which we have to show, i.e., $(a \star b) \cdot (\mathsf{s}_a \bowtie \mathsf{s}_b) \leq a \star b$ and $(\mathsf{s}_a \bowtie \mathsf{s}_b) \cdot (a \star b) \leq a \star b$ for $\star \in \{\&, \lhd\!\otimes, \otimes\!\rhd, \otimes\}$ are very analogous, hence we omit their proofs. □

Finally, the intersection and union preferences are directly translated to the algebraic setting by $a \blacklozenge b =_{df} a \sqcap b$ and $a + b$ where the union preference notationally coincides with the union in the abstract relation algebra. Again, unless otherwise specified, we have $\mathsf{s}_{a\star b} = \mathsf{s}_a \bowtie \mathsf{s}_b$ for $\star \in \{\blacklozenge, +\}$.

3.3 Maximal Element Algebra

Up to here, the major part of this chapter was a redefinition to adapt the database preferences to the abstract setting and made more concise definitions possible. In the following sections we will exhibit the strengths of the algebraic approach when proving theorems around the determination of maximal elements.

3.3.1 The Maximum Operator and its Properties

Primarily we redefine the maximum operator from Definition 2.1.11.

Definition 3.3.1 (maximal elements)**.** The *best* or *maximal* objects w.r.t. element $a :: T^2$ and a test $p :: T$ are represented by the test

$$a \rhd p =_{df} p - |a\rangle p \ .$$

This definition is also given in [DMS06] in different notation. An analogous formulation where tests are encoded as vectors, i.e., right-universal relations, can be found in [SS93].

In the following we show a number of useful basic properties of the $\rhd$ operator. The test r will always represent a finite data set. For a subset of tuples $p \leq r$ we always have $(r \cdot a \cdot r) \rhd p = a \rhd p$, by calculating

$$\begin{array}{ll} & (r \cdot a \cdot r) \rhd p \\ = & \quad \{\!\![\text{ definition }]\!\!\} \\ & p - |r \cdot a \cdot r\rangle p \\ = & \quad \{\!\![\text{ diamond properties }]\!\!\} \\ & p - r \cdot |a\rangle (r \cdot p) \\ = & \quad \{\!\![\ p \leq r \text{ and De Morgan }]\!\!\} \\ & p \cdot (\neg r + \neg |a\rangle p) \\ = & \quad \{\!\![\ p \leq r \text{ and } r \cdot \neg r = 0_a \ ]\!\!\} \\ & p - |a\rangle p = a \rhd p \ . \end{array}$$

The restriction to r will be done, using our subtype definition, by requiring $a :: T[r]^2$. This means that $a = 1_{T[r]} \cdot a \cdot 1_{T[r]}$ holds, or equivalently, $a = r \cdot a \cdot r$.

The following lemma, adapted from [MR15], collects a number of useful properties around the maximum operator.

Lemma 3.3.2 (properties of maximum)**.** *Let $a, b :: T^2$, $r :: T$ and $p \leq r$. Then the following holds:*

1. *The maximum operator is a contraction:*

$$a \rhd p \leq p \ .$$

2. *The property of being an upper estimate p with $p \leq r$ of $a \rhd r$ is invariant under the maximum operator:*

$$a \rhd r \leq p \Leftrightarrow a \rhd r \leq a \rhd p \ .$$

3. *Maximum w.r.t. the same relation is idempotent:*

$$a \rhd (a \rhd r) = a \rhd r \ .$$

4. The maximum operator over a union in the first argument distributes to an intersection of maxima:

$$(a + b) \rhd p = (a \rhd p) \cdot (b \rhd p) \ .$$

5. The maximum operator is antitone in the first argument:

$$b \leq a \ \Rightarrow \ a \rhd p \leq b \rhd p \ .$$

Proof. 1. $a \rhd p$
$= \quad \{\![$ definitions of $\rhd$ and $-$ $]\!\}$
$p \cdot \neg |a\rangle p$
$\leq \quad \{\![$ property of intersection $]\!\}$
p .

2. $a \rhd r \leq a \rhd p$
$\Leftrightarrow \quad \{\![$ definition of $\rhd$ $]\!\}$
$r - |a\rangle r \leq p - |a\rangle p$
$\Leftrightarrow \quad \{\![\ p \leq r \ \Rightarrow \ p \cdot r = p \ ]\!\}$
$r - |a\rangle r \leq p \cdot r - |a\rangle p$
$\Leftrightarrow \quad \{\![$ definition of $-$ and universal property of intersection $]\!\}$
$r - |a\rangle r \leq p \ \wedge \ r - |a\rangle r \leq r - |a\rangle p$
$\Leftrightarrow \quad \{\![$ second conjunct true by $p \leq r$ and isotony of the diamond $]\!\}$
$r - |a\rangle r \leq p$
$\Leftrightarrow \quad \{\![$ definition of $\rhd$ $]\!\}$
$a \rhd r \leq p$.

3. $a \rhd (a \rhd p)$
$= \quad \{\![$ definition of $\rhd$ $]\!\}$
$(p - |a\rangle p) - |a\rangle (p - |a\rangle)$
$= \quad \{\![$ property of difference $]\!\}$
$p - (|a\rangle p + |a\rangle (p - |a\rangle))$
$= \quad \{\![$ distributivity of $|\rangle$ $]\!\}$
$p - |a\rangle (p + (p - |a\rangle))$
$= \quad \{\![$ since $p - |a\rangle \leq p$ $]\!\}$
$p - |a\rangle p$
$= \quad \{\![$ definition of $\rhd$ $]\!\}$
$a \rhd p$.

4. $(a + b) \rhd p$
$= \quad \{\![$ definition of $\rhd$ $]\!\}$
$p - |a + b\rangle p$
$= \quad \{\![$ distributivity of $|\rangle$ $]\!\}$
$p - (|a\rangle p + |b\rangle p)$
$= \quad \{\![$ property of difference $]\!\}$

$$
\begin{aligned}
& (p - |a\rangle p) \cdot (p - |b\rangle p) \\
= \quad & \{\![\text{ definition of } \rhd \ ]\!\} \\
& (a \rhd p) \cdot (b \rhd p) \ .
\end{aligned}
$$

5. W.l.o.g. we assume $b \leq a$, i.e., $b + a = a$.

$$
\begin{aligned}
& a \rhd p \\
= \quad & \{\![\text{ assumption }]\!\} \\
& (b + a) \rhd p \\
= \quad & \{\![\text{ previous property }]\!\} \\
& (b \rhd p) \cdot (a \rhd p) \\
\leq \quad & \{\![\text{ property of intersection }]\!\} \\
& b \rhd p \ .
\end{aligned}
$$

□

The following lemma from [MR15] shows, how the maximum operator behaves on joined preferences. This will help us to study the maxima w.r.t. complex preferences.

Lemma 3.3.3. *For $a :: T_a^2, p :: T_a$ and $b :: T_b^2, q :: T_b$ with $T_a \cap T_b = \varnothing$ we have*

$$(a \bowtie b) \rhd (p \bowtie q) \ = \ (a \rhd p) \bowtie q + p \bowtie (b \rhd q) \ .$$

Proof. At first we show that for $r :: T_a, s :: T_b$ the following holds:

$$(p \bowtie q) - (r \bowtie s) \ = \ (p - r) \bowtie q \ + \ p \bowtie (q - s) \ .$$

We calculate

$$
\begin{aligned}
& (p \bowtie q) - (r \bowtie s) \\
= \quad & \{\![\text{ Lemma 3.1.7 }]\!\} \\
& (p \bowtie q) \cdot (\neg r \bowtie 1_b \ + \ 1_a \bowtie \neg s) \\
= \quad & \{\![\cdot \text{ distributes over } +, \text{ exchange law for } \cdot \text{ and } \bowtie \ ]\!\} \\
& (p \cdot \neg r) \bowtie (q \cdot 1_b) \ + \ (p \cdot 1_a) \bowtie (q \cdot \neg s) \\
= \quad & \{\![\text{ test properties }]\!\} \\
& (p - r) \bowtie q \ + \ p \bowtie (q - s) \ .
\end{aligned}
$$

Hence, by the equation above, the definitions and the distributivity of the diamond over $\bowtie$,

$$
\begin{aligned}
& (a \bowtie b) \rhd (p \bowtie q) \\
= \quad & (p \bowtie q) - |a \bowtie b\rangle (p \bowtie q) \\
= \quad & (p \bowtie q) - (|a\rangle p \bowtie |b\rangle q) \\
= \quad & (p - |a\rangle p) \bowtie q + p \bowtie (q - |b\rangle q) \\
= \quad & (a \rhd p) \bowtie q + p \bowtie (b \rhd q) \ .
\end{aligned}
$$

□

As prioritisation and Pareto composition are defined as sums of joins, we can now apply this result to the maximal sets w.r.t. those preferences.

Lemma 3.3.4. *For $a :: T_a^2, p :: T_a$ and $b :: T_b^2, q :: T_b$ with $T_a \cap T_b = \varnothing$ we have*

$$\begin{aligned}(a \mathbin{\triangleleft\!\otimes} b) \rhd (p \bowtie q) &= (a \rhd p) \bowtie q \ , \\ (a \mathbin{\otimes\!\triangleright} b) \rhd (p \bowtie q) &= p \bowtie (b \rhd q) \ , \\ (a \otimes b) \rhd (p \bowtie q) &= (a \rhd p) \bowtie (b \rhd q) \ , \\ (a \,\&\, b) \rhd (p \bowtie q) &= (a \rhd p) \bowtie (b \rhd q) \ .\end{aligned}$$

Proof.

- For $a \mathbin{\triangleleft\!\otimes} b$ we calculate

$$\begin{aligned}& (a \mathbin{\triangleleft\!\otimes} b) \rhd (p \bowtie q) \\ = \quad & \{\![\text{ definition of } a \mathbin{\triangleleft\!\otimes} b \ ]\!\} \\ & (a \bowtie (b + \mathsf{s}_b)) \rhd (p \bowtie q) \\ = \quad & \{\![\text{ Lemma 3.3.3 }]\!\} \\ & (a \rhd p) \bowtie q \ + \ p \bowtie ((b + \mathsf{s}_b) \rhd q) \\ = \quad & \{\![\text{ Lemma 3.3.2.4 }]\!\} \\ & (a \rhd p) \bowtie q \ + \ p \bowtie (b \rhd q) \cdot (\mathsf{s}_b \rhd q) \\ = \quad & \{\![\text{ by } 1_b \leq \mathsf{s}_b \text{ we get } \mathsf{s}_b \rhd q = q - \lvert \mathsf{s}_b \rangle q \leq q - q = 0_b \ ]\!\} \\ & (a \rhd p) \bowtie q \ + \ p \bowtie 0_b \\ = \quad & \{\![\bowtie \text{ is 0-strict }]\!\} \\ & (a \rhd p) \bowtie q \ .\end{aligned}$$

- For $a \mathbin{\otimes\!\triangleright} b$ the calculation is completely analogous.
- For $a \otimes b$ we get

$$\begin{aligned}& (a \otimes b) \rhd (p \bowtie q) \\ = \quad & \{\![\text{ definition }]\!\} \\ & (a \mathbin{\triangleleft\!\otimes} b + a \mathbin{\otimes\!\triangleright} b) \rhd (p \bowtie q) \\ = \quad & \{\![\text{ Lemma 3.3.2.4 }]\!\} \\ & (a \mathbin{\triangleleft\!\otimes} b \rhd (p \bowtie q)) \cdot (a \mathbin{\otimes\!\triangleright} b \rhd (p \bowtie q)) \\ = \quad & \{\![\text{ parts shown above }]\!\} \\ & ((a \rhd p) \bowtie q) \cdot (p \bowtie (b \rhd q)) \\ = \quad & \{\![\text{ exchange law for } \bowtie \text{ and } \cdot \ ]\!\} \\ & (p \cdot (a \rhd p)) \bowtie (q \cdot (b \rhd q)) \\ = \quad & \{\![\text{ by Lemma 3.3.2.1 we have } a \rhd s \leq s \ ]\!\} \\ & (a \rhd p) \bowtie (b \rhd q) \ .\end{aligned}$$

- Finally we show the claim for $a \,\&\, b$:

$$\begin{aligned}& (a \,\&\, b) \rhd (p \bowtie q) \\ = \quad & \{\![\text{ definition }]\!\} \\ & (a \bowtie \top_b + \mathsf{s}_a \bowtie b) \rhd (p \bowtie q) \\ = \quad & \{\![\text{ Lemma 3.3.2.4 }]\!\} \\ & ((a \bowtie \top_b) \rhd (p \bowtie q)) \cdot ((\mathsf{s}_a \bowtie b) \rhd (p \bowtie q))\end{aligned}$$

$$
\begin{aligned}
&= \quad \{\![\text{ Lemma 3.3.3 for both factors }]\!\} \\
&\quad ((a \rhd p) \bowtie q \;+\; p \bowtie (\top_b \rhd q)) \cdot ((\mathsf{s}_a \rhd p) \bowtie q \;+\; p \bowtie (b \rhd q)) \\
&= \quad \{\![\text{ definition of } \rhd \;]\!\} \\
&\quad ((a \rhd p) \bowtie q \;+\; p \bowtie (q - |\top_b\rangle q)) \cdot ((p - |\mathsf{s}_a\rangle p) \bowtie q \;+\; p \bowtie (b \rhd q)) \\
&= \quad \{\![\; |\top_b\rangle q = 1_b \; (\top_b \text{ is greatest element}) \text{ and } p \leq |\mathsf{s}_a\rangle p \; (\mathsf{s}_a \text{ is reflexive}) \;]\!\} \\
&\quad ((a \rhd p) \bowtie q \;+\; p \bowtie 0_b) \cdot (0_a \bowtie q \;+\; p \bowtie (b \rhd q)) \\
&= \quad \{\![\; \bowtie \text{ is 0-strict }]\!\} \\
&\quad ((a \rhd p) \bowtie q) \cdot (p \bowtie (b \rhd q)) \\
&= \quad \{\![\text{ exchange law for } \bowtie \text{ and } \cdot,\; c \rhd s \leq s \text{ (cf. proof for } a \ll\!\otimes b) \;]\!\} \\
&\quad (a \rhd p) \bowtie (b \rhd q) \; .
\end{aligned}
$$

□

This means, that the maxima of $\otimes$ and $\&$ are identical on joined data sets $p \bowtie q$, where p and q have disjoint types. Such data sets are isomorphic to Cartesian products. In Example 3.3.7 we will show that the maxima of $a \otimes b$ and $a \,\&\, b$ are different on general data sets. These general data sets are arbitrary sums of joins.

3.3.2 Examples for the Maximum Operator

In the following we give a first example to show how we algebraically reason about maximal elements for preferences.

Example 3.3.5. Let $a :: T[r]^2$ be a preference relation where r is a data set and suppose $p_1, p_2 :: T[r]$ are tests that form a disjoint decomposition of $1_{T[r]} = r$, i.e., $p_1 + p_2 = r$ and $p_1 \cdot p_2 = 0$. Hence p_1 and p_2 are complements. Assume that all elements in p_2 are better than all elements in p_1, i.e.,

$$|a\rangle p_2 = p_1 \;, \quad |a\rangle p_1 = 0_a \; .$$

We show that p_2 represents the maximal elements, i.e., $p_2 = a \rhd r$:

$$
\begin{aligned}
& a \rhd r \\
= \quad & \{\![\text{ definition of } \rhd \text{ and neutrality of } r = 1_{T[r]} \;]\!\} \\
& \neg |a\rangle r \\
= \quad & \{\![\; p_1 + p_2 = r \;]\!\} \\
& \neg(|a\rangle(p_1 + p_2)) \\
= \quad & \{\![\text{ distributivity of diamond }]\!\} \\
& \neg(|a\rangle p_1 + |a\rangle p_2) \\
= \quad & \{\![\text{ assumptions on } a \;]\!\} \\
& \neg p_1 \\
= \quad & \{\![\; p_1 \text{ and } p_2 \text{ are complements }]\!\} \\
& p_2 \; .
\end{aligned}
$$

By this tiny example one can see how the maximality operator works in general, as one can always decompose a data set r into tests representing the non-maximal (here p_1) and the maximal (here p_2) elements, where p_1 and p_2 are disjoint.

To make the following examples more illustrative, we will use explicitly given data sets and tuples in the following. We will also explicitly define preferences by using the constructors from Section 2.2. To avoid notational overhead we will use a bit "sloppy" notation in the examples, slightly deviating from the pure algebraic notation. For types

$A, B, ...$ with the same numerical domains $D_A = D_B = \{1, 2, ...\}$ we will define for each example which operand of the $\bowtie$ operator belongs to which type. For example, we say that in a expression $\alpha \bowtie \beta$ with $\alpha, \beta \in D_A$ $(= D_B)$, that the left hand side α is of type A and the right hand side of type B. This makes the $\bowtie$ operator notationally non-commutative. The domain value 1 shall not be confused with the identity 1_A for A-typed elements. We illustrate this notation in the following example.

Example 3.3.6. Let A, B be types with $D_A = D_B = \{1, 2\}$. To express a tuple t with an A-value of 1 and a B-value of 2, the formal rigorous way is

$$x = 1 :: A,\ y = 2 :: B,\ t = x \bowtie y\ .$$

To shorten this notation, we use the convention that in each term $x \bowtie y$ we have $x :: A$ and $y :: B$. Then we can express such a t by $t = 1 \bowtie 2$.

Subsequently, we show a first example of different complex preference selections on a concrete data set.

Example 3.3.7. Assume attributes A, B with domains $D_A = D_B = \mathbb{N}$. Assume a data set $r = 1 \bowtie 2 + 2 \bowtie 1 + 2 \bowtie 2$ where the first argument of $\bowtie$ corresponds to type A and the second to B. We define the following three preferences

$$\begin{aligned} a &= \mathsf{low}(A) \,\&\, \mathsf{low}(B)\ , \\ b &= \mathsf{low}(A) \otimes \mathsf{low}(B)\ , \\ c &= \mathsf{is\_true}(A = 2) \,\&\, \mathsf{low}(B)\ . \end{aligned}$$

According to the definition of these preferences and the maximum operator we get

$$\begin{aligned} a \rhd r &= 1 \bowtie 2\ , \\ b \rhd r &= 1 \bowtie 2 \ +\ 2 \bowtie 1\ , \\ c \rhd r &= 2 \bowtie 1\ . \end{aligned}$$

3.4 Prefilters

The most important application of this theoretical framework of database preferences is the optimization of the maxima calculation, i.e., the processing of a database preference query. A common strategy for an optimized query processing consists in a stepwise computation of the maxima set. This means, we first calculate a superset of the desired output and search for the maxima within this superset. For a given preference a and a data set r, we aim to find a preference b for which the maxima set on r is a superset of the maxima w.r.t. a. Then it suffices to consider this superset when finally calculating the a-maxima in r. This yields an optimization as long as the calculation costs of both steps in $a \rhd (b \rhd r)$ are less than directly calculating $a \rhd r$.

This concept is introduced as *prefilters* in [End11]. Subsequently, we introduce the theory around prefilters, where we adapt from [MRE12]. Later on, especially in Section 4.3, we will see how the computation costs can be effectively reduced due to the use of a prefilter.

3.4.1 Definition of Prefilters

At first, we give a formal definition of the concept.

Definition 3.4.1 (prefilter)**.** Assume $a, b :: T^2$ and a data set $r :: T$. We say that b is a *prefilter on* r for a by defining

$$b \,\mathsf{pref}_r\, a \quad \Leftrightarrow_{df} \quad a \rhd r = a \rhd (b \rhd r) \ .$$

By Lemma 3.3.2.2 we have $a \,\mathsf{pref}_r\, a$ for all a.

Next to the property that our semiring elements are preference relations, we will need one more requirement in the following. The concept of *normality* for a preference a introduced in [MRE12] requires that every non-maximal object is dominated by a maximal object. We specialize this to the concept of r-normality in the following.

Definition 3.4.2 (r-normality)**.** We call $a :: T^2$ *r-normal* (with $r :: T$) if

$$\forall\, p \leq r : |a\rangle p \leq |a\rangle(a \rhd p) \ .$$

By $a \rhd p \leq p$ and isotony of diamond this strengthens to

$$\forall\, p \leq r : |a\rangle p = |a\rangle(a \rhd p) \ .$$

Note that every element $a :: T[r]^2$ is trivially r-normal if r is finite. For non-finite data sets, normality requires the absence of infinite a-chains. An extensive discussion of this topic is given in [MR15].

3.4.2 Properties of Prefilters

In the subsequent theorem, adapted from [MR15], we consider subset preferences and point out a connection to prefilters.

Theorem 3.4.3. *Let $a, b :: T^2$ be preferences and $r :: T$ with $b \rhd r \leq a \rhd r$.*

1. *If b is r-normal, then a is a prefilter for b on r, i.e.,*

$$b \rhd (a \rhd r) = b \rhd r \ .$$

2. *If additionally $b \rhd q \leq a \rhd q$ holds for all $q \leq r$, then for all $p \leq r$ we also have*

$$a \rhd (b \rhd p) = b \rhd p \ .$$

Proof. 1. First we show "≤":

$$\begin{array}{ll} & b \rhd (a \rhd r) \\ = & \quad \{\![\text{ definitions }]\!\} \\ & (r - |a\rangle r) - |b\rangle(a \rhd r) \\ \leq & \quad \{\![\text{ definition of } -]\!\} \\ & r - |b\rangle(a \rhd r) \\ \leq & \quad \{\![\text{ assumption and isotony of diamond }]\!\} \\ & r - |b\rangle(b \rhd r) \\ = & \quad \{\![\ b \text{ is } r\text{-normal }]\!\} \\ & r - |b\rangle r \\ = & \ b \rhd r \ . \end{array}$$

Then we show "≥":

$$
\begin{aligned}
& b \rhd (a \rhd r) \\
= \quad & \{\![\text{ definitions }]\!\} \\
& (r - |a\rangle r) - |b\rangle (r - |a\rangle r) \\
\geq \quad & \{\![\text{ isotony of diamond }]\!\} \\
& (r - |a\rangle r) - |b\rangle r \\
\geq \quad & \{\![\text{ assumption and isotony of diamond }]\!\} \\
& (r - |b\rangle r) - |b\rangle r \\
= \quad & \{\![\text{ tests are idempotent, definitions }]\!\} \\
& b \rhd r \ .
\end{aligned}
$$

2. "≤": We have $a \rhd (b \rhd p) \leq b \rhd p$ by Lemma 3.3.2.1.
   "≥": We use the assumption for $q := b \rhd p$, which is allowed because $p \leq r$ and Lemma 3.3.2.1 implies $q \leq r$. This yields $b \rhd (b \rhd p) \leq a \rhd (b \rhd p)$ and Lemma 3.3.2.2 shows the claim. □

Note that $a \leq b$ is a sufficient premise for $\forall q :: T : b \rhd q \leq a \rhd q$ by Lemma 3.3.2.5. Hence, by the above theorem, $a \leq b$ implies that a is a prefilter for b on any data set r. The premise of b being r-normal is clearly fulfilled, because data sets are always finite in practice. Particularly this implies

$$a \,\mathsf{pref}_r\, (a + b) \ ,$$

if $(a + b)$ is r-normal, because we have $a \leq a + b$ by definition.

The above theorem also leads to a simpler characterisation of prefilters which we state subsequently. An immediate consequence of this characterisation is, that prefilters can be nested, i.e., the prefilter property is transitive. Intuitively spoken, this corresponds to a chain of prefilters, from a coarse grained one to a fine grained prefilter.

Corollary 3.4.4. *Let $a, b :: T^2$ be preferences where b is r-normal with $r :: T$.*

1. *Prefilters can be characterised by the subsumption order on their maximal sets.*

$$a \,\mathsf{pref}_r\, b \iff b \rhd r \leq a \rhd r \ .$$

2. *The "is prefilter of" relation is transitive, i.e., we have*

$$a \,\mathsf{pref}_r\, b \wedge b \,\mathsf{pref}_r\, c \implies a \,\mathsf{pref}_r\, c \ .$$

Proof. 1. We split the claim in two parts:
   "⇒" was shown in Theorem 3.4.3.1.
   "⇐" follows from Lemma 3.3.2.1 with $p = a \rhd r$.

2. From Part 1 we get that the claim is equivalent to

$$b \rhd r \leq a \rhd r \ \wedge \ c \rhd r \leq b \rhd r \ \Rightarrow \ c \rhd r \leq a \rhd r \ ,$$

   which follows immediately from the transitivity of the subsumption order. □

We consider another property of the maximum operator, which is somehow related to prefilters. Given a preference $a :: T^2$, then the union of its maxima in p and q (with $p, q :: T$) acts like a prefilter on $p + q$. Formally we have

$$a \rhd (p + q) = a \rhd (a \rhd p + a \rhd q) \ ,$$

which is shown in [MR15], Theorem 4.9, in a point-free way. A point-wise proof was already given in [HK05], Theorem L3. This law plays an important role for the distributed computation of maxima, where $p+q$ represents a data set distributed into two parts. This generalizes straight forward to n parts, i.e., for data sets $p_i :: T$ with $i = 1, ..., n$ we have

$$a \rhd \sum_{i=1}^{n} p_i = a \rhd \left(\sum_{i=1}^{n} (a \rhd p_i) \right) \ .$$

3.5 Layered Preferences and Regularisation

In Definition 3.2.1 we defined layered preferences, mathematically strict weak orders. They are not closed under the Pareto operator. For a layered preference, all tuples are either in a better-than-relation or in the same SV equivalence class. This is not the case for a Pareto preference; in general there are some tuples within a data set which are incomparable w.r.t. a given Pareto preference. This is particularly an issue, if the Pareto preference is followed by an prioritization where we get counterintuitive results in some cases. This problem is extensively studied in [MR12b], and this section mainly adapts therefrom.

Before going into details, we have to calculate maximal elements for a preference on data sets having a larger type than the preference relation. To this end we define a modified maximum operator.

Definition 3.5.1 (maximum operator for larger types)**.** Let $a :: T^2$ be a preference and $r :: T'$ a data set with $T \subseteq T'$. We define

$$a \mathbin{\widehat{\rhd}} r =_{df} (a \bowtie \top_{T' \backslash T}) \rhd r \ .$$

This operator simply picks the a-maxima in r while ignoring the additional columns $T' \backslash T$ of the data set.

We illustrate the consequences of a combination of $\otimes$ and $\&$ in the following example. We use the same notation as in Example 3.3.7.

Example 3.5.2. Assume attributes A, B, C with domains $D_A = D_B = D_C = \mathbb{N}$. Assume a data set

$$\begin{aligned} r = {} & 1 \bowtie 2 \bowtie 1 + \\ & 2 \bowtie 1 \bowtie 2 + \\ & 2 \bowtie 2 \bowtie 3 \ , \end{aligned}$$

with elements of type $x :: A, y :: B, z :: C$ in every term $x \bowtie y \bowtie z$. We define the preference

$$a = \underbrace{(\mathsf{low}(A) \otimes \mathsf{low}(B))}_{= b} \,\&\, \mathsf{low}(C) \ .$$

The optimal tuples in r w.r.t. b are

$$b \mathbin{\widehat{\rhd}} r = 1 \bowtie 2 \bowtie 1 \ + \ 2 \bowtie 1 \bowtie 2 \ .$$

According to the preference $\mathsf{low}(C)$ the first tuple is better. But for the prioritisation $a = b \,\&\, \mathsf{low}(C)$ we get the same maxima as for b, i.e., $a \rhd r = b \mathbin{\widehat{\rhd}} r$, because both tuples are incomparable w.r.t. $\mathsf{low}(A) \otimes \mathsf{low}(B)$. This means that $\ldots \,\&\, \mathsf{low}(C)$ does not change the result, although one would intuitively expect that $1 \bowtie 2 \bowtie 1$ is better than $2 \bowtie 1 \bowtie 2$ w.r.t. a, because we have $(2\,\mathsf{low}(C)\,1)$ regarding type C.

3.5.1 Layered Preference Transformation

As a remedy we suggest a new kind of prioritisation operator where the left hand side argument is transformed into a layered preference. The strategy for this transformation for a given preference a is as follows:

- In a data set r we take the maxima, i.e., $a \rhd r$ and define them as layer-0 elements.
- We remove these maxima from the data set and take the maxima from the remainder, i.e., $a \rhd (r - (a \rhd r))$. They are called layer-1 elements.
- The tuples of layer-k are determined by calculating the a-maxima in the difference between the data set and the layer-$(k-1)$ tuples. This process is iterated until no tuples are left.

The concept of layer-i tuples in the context of preferences was originally introduced in [Cho03] under the term "iterated preferences". In the following we give a precise algebraic definition of this transformation, and prove some properties. Especially we will show that the construction indeed leads to a layered preference and that we get a "layered approximation" in the sense that all better-than-relations from the original preference are preserved.

Definition 3.5.3 (layer-i elements)**.** Let $a :: T^2$ be a preference, and $r :: T$ a data set. For $i = 0, 1, 2, \ldots$ we define the tests q_i and r_i characterising the layer-i elements and the remainders, respectively:

$$q_i =_{df} a \rhd r_i \text{ where } r_i =_{df} r - \sum_{j=0}^{i-1} q_j \ .$$

By convention, the empty sum is 0_a, hence we have $r_1 = r$.

A mnemonic for the q_i is that the letter "b" for "best", rotated by 180° becomes a "q". This matches our convention that $a, b, c, \ldots$ are used for preferences and p, q, r, s for sets of tuples.

In the following Lemma from [MR15] we derive a closed formula for the r_i. In the following $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ represents the natural numbers including 0.

Lemma 3.5.4 (closed formula for layer-i elements)**.** *For $i \in \mathbb{N}_0$ we have the following properties:*

1. $(r \cdot a)^{i+1} \leq (r \cdot a)^i$ *for* $i \geq 1$,
2. $|(r \cdot a)^{i+1}\rangle r \leq |(r \cdot a)^i\rangle r$,
3. $r_i = |(r \cdot a)^i\rangle r$.

Proof. 1. By transitivity of a we have

$$(r \cdot a)^2 = r \cdot a \cdot r \cdot a \leq r \cdot a \cdot a \leq r \cdot a = (r \cdot a)^1 \ ,$$

which implies transitivity of $(r \cdot a)$. Iterated application of transitivity shows the claim.

2. For $i = 0$ we obtain by a diamond property

$$|(r \cdot a)^1\rangle r = |r \cdot a\rangle r = r \cdot |a\rangle r \leq r = |(r \cdot a)^0\rangle r \ .$$

For $i > 0$ the claim is immediate from Part 1 and isotony of diamond.

3. We perform again an induction on i.
   - $i = 0$: $|(r \cdot a)^0\rangle r = |1\rangle r = r$.
   - $i \to i + 1$: Assume $r_i = |(r \cdot a)^i\rangle r$.

$$\begin{aligned}
& r_{i+1} \\
= \quad & \{\!\![\text{ definitions }]\!\!\} \\
& r - \sum_{j=0}^{i} q_j \\
= \quad & \{\!\![\text{ distributivity }]\!\!\} \\
& \left(r - \sum_{j=0}^{i-1} q_j\right) \cdot (r - q_i) \\
= \quad & \{\!\![\text{ definition of } r_i \text{ and } r_i \leq r \ ]\!\!\} \\
& r_i - q_i \\
= \quad & \{\!\![\text{ definition } q_i \ ]\!\!\} \\
& r_i - (a \rhd r_i) \\
= \quad & \{\!\![\text{ definition } \rhd \ ]\!\!\} \\
& r_i - (r_i - |a\rangle r_i) \\
= \quad & \{\!\![\text{ definition of } -, \text{ De Morgan }]\!\!\} \\
& r_i \cdot (\neg r_i + |a\rangle r_i) \\
= \quad & \{\!\![\text{ distributivity, } p \cdot \neg p = 0 \ ]\!\!\} \\
& r_i \cdot |a\rangle r_i \\
= \quad & \{\!\![\ r_i \leq r \text{ by definition }]\!\!\} \\
& r_i \cdot r \cdot |a\rangle r_i \\
= \quad & \{\!\![\text{ diamond property }]\!\!\} \\
& r_i \cdot |r \cdot a\rangle r_i \\
= \quad & \{\!\![\text{ induction hypothesis }]\!\!\} \\
& \big(|(r \cdot a)^i\rangle r\big) \cdot \big(|r \cdot a\rangle |(r \cdot a)^i\rangle r\big) \\
= \quad & \{\!\![\text{ diamond property, definition of powers }]\!\!\} \\
& \big(|(r \cdot a)^i\rangle r\big) \cdot \big(|(r \cdot a)^{i+1}\rangle r\big) \\
= \quad & \{\!\![\text{ Part 2 }]\!\!\} \\
& |(r \cdot a)^{i+1}\rangle r \ .
\end{aligned}$$

□

In the following corollary from [MR15] we derive some properties for the q_i and r_i. These properties turn out to be useful for defining the "layered approximation" which we have sketched above.

Lemma 3.5.5. *Assume q_i, r_j as in Definition 3.5.3. We have:*

1. *The r_i are decreasing in i, i.e., $r_0 \geq r_1 \geq r_2 \geq \ldots$.*
2. *The q_i are pairwise disjoint, i.e., for $i \neq j$ we have $q_i \cdot q_j = 0_a$.*

3. *Let r be finite, i.e., assume that there do not exist infinitely many disjoint $p_i \neq 0$ with $\sum_i p_i = r$. Then the calculation of the r_i becomes stationary, i.e., there exists an $N \in \mathbb{N}_0$ with $N = \max\{k \in \mathbb{N}_0 \mid r_k \neq 0_a\}$.*

4. *The q_i cover r, i.e., $\sum_{i=0}^{N} q_i = r$.*

5. *For $i \leq j$ we have $q_i \cdot a \cdot q_j = 0_a$.*

Proof. 1. Immediate from Lemma 3.5.4.

2. Let w.l.o.g. $j \geq i+1$. It follows:

$$
\begin{array}{ll}
 & q_i \cdot q_j \\
= & \quad \{\![\text{ definition of } r_i \text{ and } \rhd \,]\!\} \\
 & (r_i - |a\rangle r_i) \cdot (r_j - |a\rangle r_j) \\
= & \quad \{\![\text{ Boolean algebra }]\!\} \\
 & r_i \cdot r_j - (|a\rangle r_i + |a\rangle r_j) \\
= & \quad \{\![\; r_j \leq r_i \text{ by (1) and } j \geq i+1, \text{ isotony of diamond }]\!\} \\
 & r_j - |a\rangle r_i \\
= & \quad \{\![\text{ Lemma 3.5.4 for } r_i, r_j \,]\!\} \\
 & |(r \cdot a)^j\rangle r - |a\rangle|(r \cdot a)^i\rangle r \\
\leq & \quad \{\![\; r \leq 1_a \,]\!\} \\
 & |(r \cdot a)^j\rangle r - r \cdot |a\rangle|(r \cdot a)^i\rangle r \\
= & \quad \{\![\text{ diamond properties }]\!\} \\
 & |(r \cdot a)^j\rangle r - |(r \cdot a)^{i+1}\rangle r \\
= & \quad \{\![\; (r \cdot a)^j \leq (r \cdot a)^{i+1} \text{ by Lemma 3.5.4.1 and } j \geq i+1 \,]\!\} \\
 & 0_a \; .
\end{array}
$$

3. By transitivity and irreflexivity of a together with the finiteness of r there are always maximal elements in non-empty sets, i.e., we have $r \neq 0_a \;\Rightarrow\; a \rhd r \neq 0_a$. Hence $r_i \neq 0_a$ implies $q_i \neq 0_a$. Additionally the q_i are pairwise disjoint by Part 2, hence r_{i+1} is strictly less (i.e., $r_{i+1} \leq r_i \wedge r_{i+1} \neq r_i$) than r_i. Induction shows that the sequence r_i is strictly decreasing for $i = 0, ..., (N+1)$ and equals 0_a for $i = (N+1), ..., \infty$.

4. Immediate from Part 3 and the definition of the q_i, since the definition of N implies $r_{N+1} = 0_a$.

5. First, we have

$$
\begin{array}{ll}
 & q_i \cdot a \cdot q_j = 0_a \\
\Leftrightarrow & \quad \{\![\; 0_a \text{ is the smallest element }]\!\} \\
 & (q_i \cdot a \cdot q_j) \leq 0_a \\
\Leftrightarrow & \quad \{\![\text{ definition of diamond }]\!\} \\
 & |q_i \cdot a\rangle q_j \leq 0_a \\
\Leftrightarrow & \quad \{\![\text{ property of diamond }]\!\} \\
 & q_i \cdot |a\rangle q_j \leq 0_a \; .
\end{array}
$$

Now,

$$
\begin{aligned}
& q_i \cdot |a\rangle q_j \\
= \quad & \{\![\text{ definition of } q_i, q_j \ ]\!\} \\
& (r_i - |a\rangle r_i) \cdot |a\rangle (r_j - |a\rangle r_j) \\
\leq \quad & \{\![\text{ isotony of diamond }]\!\} \\
& (r_i - |a\rangle r_i) \cdot |a\rangle r_j \\
\leq \quad & \{\![\ i \leq j, \text{ hence } r_j \leq r_i \text{ by Part 1 }]\!\} \\
& (r_i - |a\rangle r_i) \cdot |a\rangle r_i \\
= \quad & \{\![\text{ Boolean algebra }]\!\} \\
& 0_a \ .
\end{aligned}
$$

□

Based on the results above we formalize the transformation of an arbitrary preference into a layered preference by defining the *induced layered preference*.

Definition 3.5.6 (induced layered preference)**.** Let $a :: T^2$ be a preference and $r :: T$ a data set. Consider the corresponding layer-i elements $q_i = a \rhd |(r \cdot a)^i\rangle r$ with $i \in \{1, ..., N\}$ and $N = \max\{k \in \mathbb{N}_0 \mid r_k \neq 0_a\}$, according to Lemma 3.5.5.3. We define relations b_{ij} with $i, j \in \{1, .., N\}$ by $b_{ij} = q_i \cdot \top_a \cdot q_j$. In the concrete model these represent universal relations between the sets q_i and q_j. With their help, the *induced layered preference* $m(a, r) :: T[r]^2$ is defined as

$$m(a, r) =_{df} \sum_{i>j} b_{ij} \ ,$$

where $T[r]$ is the sub-type of T with identity r and greatest element $r \cdot \top_T \cdot r$.

Because of the summation over $i > j$ the less preferred elements w.r.t. a are $m(a, r)$-related to the more preferred elements, where the elements with lower layer numbers are better than those with higher layer numbers.

A corresponding SV relation $\mathsf{s}_{m(a,r)} :: T[r]^2$ is defined by

$$\mathsf{s}_{m(a,r)} =_{df} \sum_i b_{ii} \ .$$

Note that the induced layered preference has the measure function $f(x) = i + 1$ for $x \leq q_i$ in the sense of Corollary 3.2.3.

We note an important property of the relations b_{ij}: by disjointness of the q_i and the Tarski rule we have

$$b_{ij} \cdot b_{kl} = \begin{cases} b_{il} & \text{if } j = k \ , \\ 0_a & \text{otherwise} \ . \end{cases} \tag{3.2}$$

Lemma 3.5.7. *The relation $m(a, r)$ is well-defined, i.e., we have:*

1. *$m(a, r)$ is a layered preference.*
2. *$\mathsf{s}_{m(a,r)}$ is an SV relation for $m(a, r)$.*

Proof. 1. Transitivity follows from the definition of $m(a, r)$ and (3.2). Again by definition of $m(a, r)$ and disjointness of q_i (cf. Lemma 3.5.5.2) we have irreflexivity. It remains to show that negative transitivity holds. Note that due to the type $T[r]^2$

of $m(a,r)$ and $\mathsf{s}_{m(a,r)}$ the complement $\overline{(...)}$ is relative to r. With this we infer $(\overline{m(a,r)})^2 \leq \overline{m(a,r)}$:

$$\left(\overline{m(a,r)}\right)^2 = \left(\sum_{i\leq j} b_{ij}\right) \cdot \left(\sum_{k\leq l} b_{kl}\right) = \sum_{i\leq j\leq l} b_{ij} \cdot b_{jl} \leq \sum_{i\leq l} b_{il} = \overline{m(a,r)} \ .$$

2. We infer that

$$\overline{m(a,r) + m(a,r)^{-1}} = \overline{\sum_{i>j} b_{ij} + \sum_{i<j} b_{ij}} = \overline{\sum_{i\neq j} b_{ij}} = \sum_{i} b_{ii} = \mathsf{s}_{m(a,r)}.$$

Together with Lemma 3.2.4 this shows the claim. □

3.5.2 Properties of the Induced Preference

Next to the soundness of the definition we formally show some other useful properties of this construction. We have claimed initially that the layered approximation should contain the original preference, which we will show in the following lemma. Additionally we show that the SV relation is restricted to tuples which are not in a better-than-relation in the original preference. We restrict the original preference to r on both sides, because the induced layered preference and the induced SV relation is only defined on the data set r, i.e., its type is a sub type $T[r]$ where T is the type of the original preference.

Lemma 3.5.8. *Let $a :: T^2$ be a preference and $r :: T$ a data set. We have:*

1. $r \cdot a \cdot r \leq m(a,r)$.
2. $\mathsf{s}_{m(a,r)} \leq r \cdot \overline{(a + a^{-1})} \cdot r$.

Proof. 1. By Lemma 3.5.5.5 we get $q_i \cdot a \cdot q_j = 0$ for $i \leq j$. This implies

$$\sum_{i\leq j} q_i \cdot a \cdot q_j = 0_a \ . \tag{3.3}$$

We use this in the following deduction:

TRUE
$\Leftrightarrow$ {[definition $\top_a$]}
$a \leq \top_a$
$\Rightarrow$ {[$q_j \cdot (...)$, $(...) \cdot q_i$, summation over $i > j$]}
$\sum_{i>j} q_i \cdot a \cdot q_j \leq \sum_{i>j} q_i \cdot \top_a \cdot q_j$
$\Leftrightarrow$ {[Equation (3.3) (additional term is 0_a), definition of b_{ij}]}
$\sum_{i>j} q_i \cdot a \cdot q_j + \sum_{i\leq j} q_i \cdot a \cdot q_j \leq \sum_{i>j} b_{ij}$
$\Leftrightarrow$ {[re-indexing of sum, definition of $m(a,r)$]}
$\sum_{i,j} q_i \cdot a \cdot q_j \leq m(a,r)$
$\Leftrightarrow$ {[distributivity and $\sum_i q_i = r$ (Lemma 3.5.5.4)]}
$r \cdot a \cdot r \leq m(a,r)$.

2. The claim is equivalent to

$$\mathsf{s}_{m(a,r)} \sqcap (r \cdot a^k \cdot r) = 0_a \text{ for } k \in \{-1, 1\} .$$

From Part 1 we obtain $r \cdot a^k \cdot r \leq m(a^k, r)$ for $k \in \{-1, 1\}$, because a^{-1} is again a preference, hence the same argument holds for it. Thus it is sufficient to prove:

$$\mathsf{s}_{m(a,r)} \sqcap m(a^k, r) = 0_a \text{ for } k \in \{-1, 1\} .$$

This follows from the definitions of $\mathsf{s}_{m(a,r)}$ and $m(a^k, r)$ and the disjointness of the q_i (Lemma 3.5.5.2). □

Note that the inequations in the previous lemma are equations if a is already a layered preference.

3.5.3 Regularised Prioritisation

In the motivating Example 3.5.2 we showed that a right hand side prioritisation cannot change the order of incomparable elements of a left hand side Pareto preference. To remedy this, we define the *regularised prioritisation*, transforming the left operand into the induced layered preference from above.

Definition 3.5.9 (regularised prioritisation)**.** Let $a :: T_a^2$, $b :: T_b^2$ be preferences and $r :: T_a \bowtie T_b$ a data set. The *regularised prioritisation* is defined by

$$a \,\&_{\text{reg}}\, b :: (T_a \bowtie T_b)^2 ,$$

$$a \,\&_{\text{reg}}\, b = m(a \bowtie \top_b, r) \,\&\, b .$$

For the SV relation we have, unless otherwise specified, the usual SV relation of the above & operation, i.e.,

$$\mathsf{s}_{a \&_{\text{reg}} b} = \mathsf{s}_{m(a \bowtie \top_b, r) \& b} = \mathsf{s}_{m(a \bowtie \top_b, r)} \bowtie \mathsf{s}_b .$$

This definition corrects Definition 6.10 from [MR15] where instead of $m(a \bowtie \top_b, r)$ the term $m(a, r)$ is used. The term $m(a, r)$ is not validly typed, as $m(x, y)$ expects parameters of types $x :: T^2$ and $y :: T$ for some T.

If one just wants to calculate $(a \,\&_{\text{reg}}\, b) \rhd r$ and the entire preference order does not matter, this expression can be converted into the calculation of nested maxima. There we use the maximum operator for larger types as defined in Definition 3.5.1.

Lemma 3.5.10. *Let $a :: T_a^2, b :: T_b^2$ and $r :: T_a \bowtie T_b$. Then we have*

$$(a \,\&_{\text{reg}}\, b) \rhd r = b \mathbin{\widehat{\rhd}} (a \mathbin{\widehat{\rhd}} r) .$$

Proof. First, by the definition of $a \,\&_{\text{reg}}\, b$ and with Lemma 3.3.2.4 we infer

$$\begin{aligned} & (a \,\&_{\text{reg}}\, b) \rhd r \\ = \;& \left(m(a \bowtie \top_b, r) \bowtie \top_b \;+\; \mathsf{s}_{m(a \bowtie \top_b, r)} \bowtie b\right) \rhd r \\ = \;& \left((m(a \bowtie \top_b, r) \bowtie \top_b) \rhd r\right) \cdot \left(\left(\mathsf{s}_{m(a \bowtie \top_b, r)} \bowtie b\right) \rhd r\right) \end{aligned}$$

Next, by definition of $m(...)$, the left operand of $\cdot$ equals $p := (a \bowtie \top_b) \rhd r$. The projection of p to the type of a equals q_0 from Definition 3.5.3, formally $q_0 = p \sqcap 1_a$. On the right hand side of $\cdot$, all tuples in q_0 are SV-equivalent w.r.t. $\mathsf{s}_{m(a \bowtie \top_b)}$ by definition. Hence among p the preference $\mathsf{s}_{m(a \bowtie \top_b, r)} \bowtie b$ picks the b-best tuples in p, i.e., $(\top_a \bowtie b) \rhd p$, or equivalently $b \mathbin{\widehat{\rhd}} p$ (cf. Definition 3.5.1). As we also have $p = a \mathbin{\widehat{\rhd}} p$ by definition, the set of tuples $(a \,\&_{\text{reg}}\, b) \rhd r$ is equivalent to $b \mathbin{\widehat{\rhd}} (a \mathbin{\widehat{\rhd}} p)$, which shows the claim. □

The above lemma shows how the regularised prioritisation can be easily implemented for usual preference queries. But if one needs the top-k preference selection, i.e., the k best elements from a data set w.r.t. a given preference, this approach fails. In this case, the q_i have to be calculated as given in Definition 3.5.3. The calculation of the q_i can be stopped if there are "enough" tuples for the given top-k query, formally if i is such that $|\sum_{j=0}^{i} q_i| \geq k$ holds.

Subsequently, we revisit Example 3.5.2 using the regularised prioritisation.

Example 3.5.11. Assume attributes A, B, C with domains $D_A = D_B = D_C = \mathbb{N}$ and a data set $r = 1 \bowtie 2 \bowtie 1 \;+\; 2 \bowtie 1 \bowtie 2 \;+\; 2 \bowtie 2 \bowtie 3$ as in Example 3.5.2. We define the preferences

$$a = (\mathsf{low}(A) \otimes \mathsf{low}(B)) \,\&\, \mathsf{low}(C) \;,$$
$$b = (\mathsf{low}(A) \otimes \mathsf{low}(B)) \,\&_{\mathrm{reg}}\, \mathsf{low}(C) \;.$$

The optimal tuples in r w.r.t. a are

$$a \rhd r = 1 \bowtie 2 \bowtie 1 \;+\; 2 \bowtie 1 \bowtie 2 \;.$$

For the preference b, these tuples are SV-equivalent w.r.t. the left hand side of $\&_{\mathrm{reg}}$. Hence the preference $\mathsf{low}(C)$ finally picks the first tuple and we get

$$b \rhd r = 1 \bowtie 2 \bowtie 1 \;.$$

For the application, the introduction of regularised prioritisation plays a major role when considering long prioritisation chains, where in one of the first parts of the chain there occurs a Pareto preference. For example the context model for a hiking tour recommender suggested in [REMK12] generates such preference terms. The experiences of this application lead primarily to a "regularised Pareto preference", where the m-transformation is called by a modified Pareto constructor. As discussed in [MR15] this modified constructor results in a non-associative operator which is theoretically undesirable. Hence we restricted our attention to the quite similar concept of a regularised prioritisation.

3.6 The Distributive Law for Complex Preferences

In the following we show the left-distributivity of & over $\otimes$. In contrast to the theorems around the maximal element algebra, this law is independent of the maximum operator. In the applications of the theory later on in this thesis, we will see that this law is very important when showing the equivalence of preference terms where both prioritization and Pareto composition are involved. We first will show a lemma with two auxiliary arguments.

Lemma 3.6.1. *Let $a :: T_a^2$ and $b, b' :: T_b^2$. Then we have:*

1. *& is left-distributive over +:*

$$a \,\&\, (b + b') = a \,\&\, b + a \,\&\, b' \;.$$

2. *The Pareto operators are idempotent:*

$$a \otimes a = a \otimes a = a \otimes a \;=\; a \;.$$

Proof. 1. We calculate:

$$
\begin{aligned}
& a \,\&\, (b + b') \\
= \quad & \{\![\text{ definition }]\!\} \\
& a \bowtie \top_b \;+\; \mathsf{s}_a \bowtie (b + b') \\
= \quad & \{\![\text{ idempotency of } +, \text{ distributivity of } \bowtie \text{ over } + \,]\!\} \\
& a \bowtie \top_b \;+\; a \bowtie \top_b \;+\; \mathsf{s}_a \bowtie b \;+\; \mathsf{s}_a \bowtie b' \\
= \quad & \{\![\text{ definition }]\!\} \\
& a \,\&\, b + a \,\&\, b' \;.
\end{aligned}
$$

2. We show the claim for $\lessdot\!\otimes$

$$
\begin{aligned}
& a \lessdot\!\otimes\, a \\
= \quad & \{\![\text{ definition }]\!\} \\
& (a + \mathsf{s}_a) \bowtie a \\
= \quad & \{\![\bowtie \text{ on the same type equals } \sqcap, \text{ Definition 3.1.6.7 }]\!\} \\
& (a + \mathsf{s}_a) \sqcap a \\
= \quad & \{\![\text{ absorption law }]\!\} \\
& a \;.
\end{aligned}
$$

For $\otimes\!\gtrdot$ and $\otimes$ analogous arguments show the claim. □

After these prerequisites we state the theorem.

Theorem 3.6.2. *For $a :: T_a, b :: T_b, c :: T_c$ we have*

$$a \,\&\, (b \star c) = (a \,\&\, b) \star (a \,\&\, c)$$

where $\star \in \{\lessdot\!\otimes, \otimes\!\gtrdot, \otimes\}$ and $\&$ is assumed as SV-preserving.

Proof. We calculate

$$
\begin{aligned}
& (a \,\&\, b) \otimes\!\gtrdot (a \,\&\, c) \\
= \quad & \{\![\text{ definition of } \otimes\!\gtrdot \,]\!\} \\
& (a \,\&\, b + \mathsf{s}_{a\&b}) \bowtie (a \,\&\, c) \\
= \quad & \{\![\;\& \text{ is assumed to be SV-preserving }]\!\} \\
& (a \,\&\, b + \mathsf{s}_a \bowtie \mathsf{s}_b) \bowtie (a \,\&\, c) \\
= \quad & \{\![\text{ definition of } \& \,]\!\} \\
& (a \bowtie \top_b + \mathsf{s}_a \bowtie b + \mathsf{s}_a \bowtie \mathsf{s}_b) \bowtie (a \bowtie \top_c + \mathsf{s}_a \bowtie c) \\
= \quad & \{\![\text{ distributivity of } \bowtie \text{ over } + \,]\!\} \\
& (a \bowtie \top_b + \mathsf{s}_a \bowtie (b + \mathsf{s}_b)) \bowtie (a \bowtie \top_c + \mathsf{s}_a \bowtie c) \\
= \quad & \{\![\text{ distributivity of } \bowtie \text{ over } + \,]\!\} \\
& a \bowtie \top_b \bowtie a \bowtie \top_c \;+\; a \bowtie \top_b \bowtie \mathsf{s}_a \bowtie c \;+ \\
& \mathsf{s}_a \bowtie (b + \mathsf{s}_b) \bowtie a \bowtie \top_c \;+\; \mathsf{s}_a \bowtie (b + \mathsf{s}_b) \bowtie \mathsf{s}_a \bowtie c \\
= \quad & \{\![\; a \bowtie a = a \text{ and } a \bowtie \mathsf{s}_a = a \sqcap \mathsf{s}_a, \text{ Definition 3.1.6.7 }]\!\} \\
& a \bowtie \top_b \bowtie \top_c \;+\; (a \sqcap \mathsf{s}_a) \bowtie \top_b \bowtie c \;+ \\
& (a \sqcap \mathsf{s}_a) \bowtie (b + \mathsf{s}_b) \bowtie \top_c \;+\; \mathsf{s}_a \bowtie (b + \mathsf{s}_b) \bowtie c \\
= \quad & \{\![\; a \sqcap \mathsf{s}_a = 0_a \text{ by Lemma 3.2.2 and 0-strictness of } \bowtie \,]\!\}
\end{aligned}
$$

$$
\begin{aligned}
& a \bowtie \top_b \bowtie \top_c + \mathsf{s}_a \bowtie (b + \mathsf{s}_b) \bowtie c \\
= \quad & \{\![\ \top_{b \bowtie c} = \top_b \bowtie \top_c, \text{ definition of } \&\ ]\!\} \\
& a \,\&\, ((b + \mathsf{s}_b) \bowtie c) \\
= \quad & \{\![\ \text{definition of } \otimes\!\!>\ ]\!\} \\
& a \,\&\, (b \otimes\!\!> c)\ .
\end{aligned}
$$

A symmetric argument holds for $<\!\!\otimes$, so that $(a\,\&)$ distributes over $<\!\!\otimes$ and $\otimes\!\!>$. Finally we obtain the distributivity of $(a\,\&)$ over $\otimes$ by:

$$
\begin{aligned}
& a \,\&\, (b \otimes c) \\
= \quad & \{\![\ \text{definition of } \otimes\ ]\!\} \\
& a \,\&\, (b <\!\!\otimes c + b \otimes\!\!> c) \\
= \quad & \{\![\ \text{distributivity of } \& \text{ over } +, \text{ Lemma 3.6.1}\ ]\!\} \\
& a \,\&\, (b <\!\!\otimes c) + a \,\&\, (b \otimes\!\!> c) \\
= \quad & \{\![\ \text{distributivity of } (a\,\&) \text{ over } <\!\!\otimes \text{ and } \otimes\!\!>, \text{ Theorem 3.6.2}\ ]\!\} \\
& (a \,\&\, b) <\!\!\otimes (a \,\&\, c) + (a \,\&\, b) \otimes\!\!> (a \,\&\, c) \\
= \quad & \{\![\ \text{definition of } \otimes\ ]\!\} \\
& (a \,\&\, b) \otimes (a \,\&\, c)\ .
\end{aligned}
$$

□

In Section 6.1 we will prove this theorem using PROVER9. The distributive law for & and ⊗ will be frequently needed for the decomposition theorems in Chapter 5.

Note that we just have left-distributivity for & over ⊗. Right-distributivity does not hold in this case in general. We will subsequently show a counterexample.

Example 3.6.3. Assume attributes A, B, C with domains $D_A = D_B = D_C = \mathbb{N}$. Assume a data set

$$
\begin{aligned}
r = {} & 1 \bowtie 2 \bowtie 1 + \\
& 2 \bowtie 1 \bowtie 2 + \\
& 1 \bowtie 1 \bowtie 3\ ,
\end{aligned}
$$

where $x :: A, y :: B, z :: C$ in every term $x \bowtie y \bowtie z$. We define the preferences

$$
\begin{aligned}
a &= (\mathsf{low}(A) \otimes \mathsf{low}(B)) \,\&\, \mathsf{low}(C)\ , \\
b &= (\mathsf{low}(A) \,\&\, \mathsf{low}(C)) \otimes (\mathsf{low}(B) \,\&\, \mathsf{low}(C))\ .
\end{aligned}
$$

The optimal tuples in r w.r.t. a and b are

$$
\begin{aligned}
a \rhd r &= 1 \bowtie 1 \bowtie 3\ , \\
b \rhd r &= r\ .
\end{aligned}
$$

The example above shows that $(a \otimes b) \,\&\, c = (a \,\&\, c) \otimes (b \,\&\, c)$ is in general wrong.

CHAPTER 4

Applications in Preference Algebra

In the previous chapter we introduced preferences as the theoretical framework for preference query languages. In this chapter we will study applications on top of them and problems motivated from practice. First, we consider the algebraic modelling of grouped preferences, where the preference evaluation is done for each partition of a grouped data set. Next, we derive an optimization for computing Pareto preferences on a data stream. Finally, we present the Scalagon algorithm for Pareto preferences, which is an example for a prefilter, as introduced in the previous chapter.

4.1 Grouped Preferences

A *grouped preference selection* primarily causes the partitioning of a data set into different disjoint groups. Next, the maximum operator w.r.t. the given preference is applied to each group separately and finally the union of these maxima over all groups is returned. In the database context, the partitioning of data sets is usually controlled with the `GROUP BY` operator, that is typically used in connection with *aggregate functions*, such as `SUM`, `COUNT` or `MAX`. As the preference selection is also a calculation of maximal elements, grouped preferences are related to aggregations in standard SQL.

The most important difference concerns the codomain of the grouping and aggregating operations. In standard SQL, aggregation functions are applied over single attributes (i.e., singleton types in our type calculus). The result only contains the aggregated values together with the grouping columns, i.e., the columns which induce the equivalence class for the partitioning. For example, `SELECT x, max(y) FROM r GROUP BY x` just returns the grouping column `x` and the aggregation result `max(y)` from the data set `r`. If there occur other columns from `r` in the projection, e.g. `SELECT x, z, max(y) FROM r GROUP BY x`, this would lead to an SQL error.

In contrast to this, the grouped preference selection returns tuples which are maximal in each group w.r.t. the preference. In the commercial implementation Exasolution Skyline the `PARTITION BY` operator is exclusively used for the grouped preference selection and not for aggregate functions. For example, the query

```
SELECT x, y, z FROM r PREFERRING high(y) PARTITION BY x
```

returns all tuples with the highest y-values in every x-group. The additional projection to z is allowed, as the tuple structure is preserved by the preference selection.

Note that in Exasolution Skyline the user can combine the `PARTITION BY` directive with a `GROUP BY` construct afterwards. This allows applying an aggregating function to the preference query result. For example, a query like

```
SELECT x, count(*) FROM r PREFERRING high(y) PARTITION BY x GROUP BY x
```

returns the number of maximal y-values in every x-group.

To construct queries with preferences similar to `PREFERRING low(y) PARTITION BY x` in standard SQL, subquery constructions are widely used in decision support queries like those in the TPC-H benchmark [TPC15]. In Theorem 6 of [ERK14] it is shown, how standard SQL subquery constructions obtaining the argument of a maximum can be simplified by such a grouped preference selection. We will not go into the details of those query rewritings within this thesis, as the algebraic approach reached its limitations when applied to this kind of inherently point-wise theorems. We did not find a simple and elegant way to algebraically model the grouping and aggregating functions from standard SQL.

But at least for the grouped preference selection, a concise algebraic formulation is possible. In this section we will show some of the optimization laws for grouped preferences, having been published in [ERK14] in a slightly different notation. The main idea is, that the grouping operator is formally a simple join operation connecting the preference with the equivalence relation for the partitioning.

The pioneering paper [BKS01] already discussed the interplay between the `GROUP BY` operator, aggregate functions and the Skyline operator (i.e., Pareto preferences). There the question has been discussed whether the Skyline operator should be processed before or after the aggregating functions. In e.g., [LYL09] the Skyline processing before the aggregation is discussed.

4.1.1 Definition

Mathematically, a grouping operation corresponds to an equivalence relation on the data set. This equivalence relation is induced by a measure function. All tuples where the grouping attribute has the same value, are in the same equivalence class. Adapted to [ERK14] we define:

Definition 4.1.1 (grouping operation)**.** Let $a :: T_a^2$ be a preference and $b :: T_b^2$ an equivalence relation, i.e., $1_T \leq b$, $b^2 \leq b$ and $b = b^{-1}$. We define the *grouped preference of a where the partitioning is subject to b* by

$$a \text{ grouping } b =_{df} a \bowtie b \ , \qquad \mathsf{S}_a \text{ grouping } b =_{df} \mathsf{S}_a \bowtie b \ .$$

For a mapping $\phi : D_T \to D$ from a type domain D_T to some other domain D we define the *grouped preference of a where the partitioning is subject to ϕ* by

$$a \text{ grouping\_by } \phi =_{df} a \text{ grouping } c \quad \text{where} \quad \forall x, y :: T : \ x \, c \, y \Leftrightarrow_{df} \phi(x) = \phi(y) \ .$$

Analogously to the definitions of base preferences in Section 2.2 we will use the notation a grouping_by *expr* where *expr* is an expression over the attributes in the type T.

To get a first intuition of the grouping operation, consider the following simple example of a grouped preference selection.

Example 4.1.2. Assume attributes A, B with domains $D_A = D_B = \mathbb{N}$ and the data set

$$r = 1 \bowtie 1 \;+\; 1 \bowtie 2 \;+\; 2 \bowtie 3 \;+\; 2 \bowtie 4 \;,$$

where $x :: A, y :: B$ in every term $x \bowtie y$. We define the preference $a = \mathsf{low}(B)$ $\mathsf{grouping\_by}$ A which is equivalent to $\mathsf{low}(B)$ $\mathsf{grouping}$ 1_A by Definition 4.1.1. For the maximum we get

$$a \rhd r = 1 \bowtie 1 \;+\; 2 \bowtie 3 \;.$$

In this example the two partitions are induced by $A = 1$ and $A = 2$, respectively.

4.1.2 Transformation Rules

In the following we will show some transformation laws for grouped preferences. We follow the ideas of [KH03, REK13, ERK14] and translate the proofs into the algebraic setting.

Lemma 4.1.3. *Let $a, b :: T_a^2$ be preferences and $c, d :: T_b^2$ equivalence relations. Nested grouping operations and complex preference operators in conjunction with* $\mathsf{grouping}$ *can be simplified as follows:*

1. $(a \;\mathsf{grouping}\; c) \;\mathsf{grouping}\; d = a \;\mathsf{grouping}\; (c \bowtie d)$
2. $(a \;\mathsf{grouping}\; c) \,\&\, b = (a \,\&\, b) \;\mathsf{grouping}\; c$
3. $(a \;\mathsf{grouping}\; c) \otimes b = (a \otimes b) \;\mathsf{grouping}\; c = a \otimes (b \;\mathsf{grouping}\; c)$
4. $(a \;\mathsf{grouping}\; c) \otimes (b \;\mathsf{grouping}\; d) = (a \otimes b) \;\mathsf{grouping}\; (c \bowtie d)$

Proof. 1. Immediately from definition and associativity of $\bowtie$.

2.
$$\begin{aligned}
& (a \;\mathsf{grouping}\; c) \,\&\, b \\
= \;& \{\![\text{ definition of } \mathsf{grouping} \text{ and } \& \;]\!\} \\
& a \bowtie c \bowtie \top_b \;+\; \mathsf{s}_a \bowtie c \bowtie b \\
= \;& \{\![\text{ associativity and commutativity of } \bowtie \;]\!\} \\
& (a \bowtie \top_b) \bowtie c \;+\; (\mathsf{s}_a \bowtie b) \bowtie c \\
= \;& \{\![\text{ distributivity of } \bowtie \text{ over } + \;]\!\} \\
& (a \bowtie \top_b \;+\; \mathsf{s}_a \bowtie b) \bowtie c \\
= \;& \{\![\text{ definition of } \& \text{ and } \mathsf{grouping} \;]\!\} \\
& (a \,\&\, b) \;\mathsf{grouping}\; c \;.
\end{aligned}$$

3. We show the first part by

$$\begin{aligned}
& (a \;\mathsf{grouping}\; c) \otimes b \\
= \;& \{\![\text{ definition }]\!\} \\
& a \bowtie c \bowtie (\mathsf{s}_b + b) \;+\; (a \bowtie c \;+\; \mathsf{s}_a \bowtie c) \bowtie b \\
= \;& \{\![\text{ distributivity of } \bowtie \text{ over } + \;]\!\} \\
& a \bowtie c \bowtie (\mathsf{s}_b + b) \;+\; (a + \mathsf{s}_a) \bowtie c \bowtie b \\
= \;& \{\![\text{ associativity/commutativity of } \bowtie \text{ and distributivity of } \bowtie \text{ over } + \;]\!\} \\
& (a \bowtie (\mathsf{s}_b + b) \;+\; (a + \mathsf{s}_a) \bowtie b) \bowtie c \\
= \;& \{\![\text{ definition }]\!\} \\
& (a \otimes b) \;\mathsf{grouping}\; c \;.
\end{aligned}$$

The second part is clear by commutativity of $\otimes$.

4. Immediate by Parts 1 and 3. □

Next, we consider the special case of a grouping operation where each group consists of a single tuple.

Lemma 4.1.4. *Let $a :: T_a^2$ be a preference and $r :: T$ a data set with $T_a \subseteq T$. If the grouping is unique for every tuple in r, i.e., equivalent to the identity on r, then the maximum w.r.t. such a grouped preference is also the identity. Formally we have*

$$(a \text{ grouping } 1_T) \rhd r = r \ .$$

Proof. a grouping $1_T = a \bowtie 1_T$

$=$ {[type decomposition of 1_T]}

$a \bowtie 1_a \bowtie 1_{T \setminus T_a}$

$=$ {[join equals meet on the same type]}

$(a \sqcap 1_a) \bowtie 1_{T \setminus T_a}$

$=$ {[$a \sqcap 1_a = 0_a$ by irreflexivity of a]}

$0_a \bowtie 1_{T \setminus T_a}$

$=$ {[$\bowtie$ is 0-strict]}

0_T .

Using this we have $(a \text{ grouping } 1_T) \rhd r = 0_T \rhd r = r - |0_T\rangle r = r$ which shows the claim. □

With the two lemmas above we have shown laws which where derived in [KH03] in a point-wise and more lengthy manner. The arguments in the proofs presented here mainly rely on algebraic properties of the join operator, as axiomatized in the join algebra, Definition 3.1.6. This underlines the usefulness of the algebraic approach.

If we consider the laws from Lemma 4.1.3 one notices that the symmetric law for Part 2, $a \,\&\, (b \text{ grouping } c) = (a \,\&\, b) \text{ grouping } c$, is not contained. This is generally wrong, as we will show in the following counterexample.

Example 4.1.5. Let A, B, C be attributes with $D_A = D_B = D_C = \mathbb{N}$ and a data set

$$r = 1 \bowtie 1 \bowtie 1 \ + \ 2 \bowtie 2 \bowtie 2 \ .$$

We get the following results:

$$(\mathsf{low}(A) \,\&\, (\mathsf{low}(B) \text{ grouping } 1_C)) \rhd r = 1 \bowtie 1 \bowtie 1 \ ,$$
$$((\mathsf{low}(A) \,\&\, \mathsf{low}(B)) \text{ grouping } 1_C) \rhd r = r \ .$$

4.2 Pareto Fronts on Streaming Data

In this section we study the dominance region of tuples w.r.t. a Pareto preference. This can be particularly applied to streaming data, i.e., a data set where continuously new tuples are added. We want to keep the Pareto front always up to date, i.e., the maximum set w.r.t. a Pareto preference. We suggest a method, where we use the dominance region of the existing tuple to perform a quick check if the new tuple belongs to the maximum set or not. For deriving this method we use the algebraic and relational approach introduced previously. In this section we follow Chapter 5 of [DGM+14].

4.2.1 Idea

Figure 4.1 shows the maximal elements w.r.t. a $\mathsf{high}(A) \otimes \mathsf{high}(B)$ preference, where A, B are the attributes corresponding to the coordinates in the diagram. The Pareto frontier, visible as the stair-shaped line in the figure, subdivides the domain into two areas:

1) The *dominance region* consisting of the maximal elements of the given data set and those tuples being dominated by the maxima,

2) the non-dominated area in the upper right, where no tuples from the data set are contained.

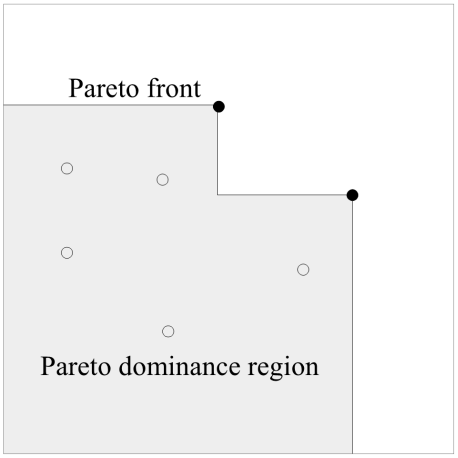


Figure 4.1: The Pareto dominance region (grey) and the Pareto front (black line). The filled circles are the maxima and the unfilled circles are dominated objects.

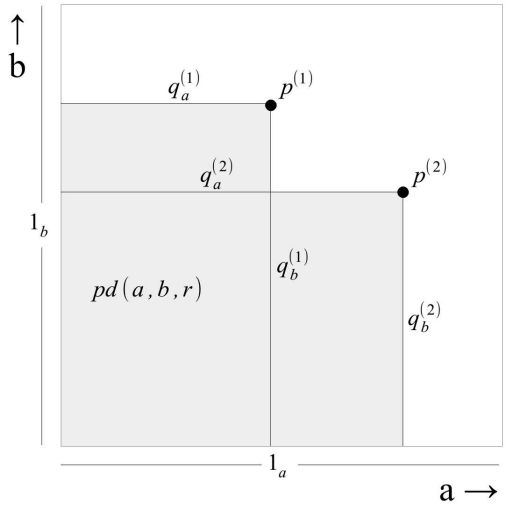


Figure 4.2: Rectangular representation of this region. The arrows indicate where elements are better w.r.t. the preferences a and b. The $p^{(i)}$ are points while the $q^{(i)}$ are areas.

For the following we will need the formal definition of a *rectangle*, corresponding to rectangular regions in the picture.

Definition 4.2.1 (rectangle)**.** A *rectangle* $r :: T_a \bowtie T_b$ is a data set which can be written as a join, i.e., there are elements $r_a :: T_a$ and $r_b :: T_b$, such that $r = r_a \bowtie r_b$.

Rectangles are isomorphic to Cartesian products. We show that the dominance region can be described by N rectangles for N tuples in the Pareto front. Its complement can be represented by $N + 1$ rectangles.

Ordering these rectangles by size (or a weighted size w.r.t. a given probability distribution estimating where new tuples will be added) paves the way for a fast calculation on which side of the Pareto front a new element would be placed, i.e., if it is dominated or not.

4.2.2 Representing the Pareto Front by Rectangles

Let $a :: T_a^2, b :: T_b^2$ be layered preferences with disjoint types T_a, T_b and let $r :: T_a \bowtie T_b$ be a data set. The maximal elements of the Pareto preference $a \otimes b$ in r are given by $(a \otimes b) \rhd r =_{df} r - \lfloor a \otimes b \rangle r$.

In the following we will assume that the SV relations of a, b are the identity, formally $\mathsf{s}_a = 1_a$ and $\mathsf{s}_b = 1_b$. If this is not the case, one always finds a mapping from $a :: T^2$ to $a' :: T'^2$ such that $1_{T'}$ is given by the equivalence classes of s_a and hence $\mathsf{s}_{a'} = 1_{T'}$. The convention of $\mathsf{s}_a = 1_a$ and $\mathsf{s}_b = 1_b$ ensures that all points in Figure 4.2 which are on the same vertical line (or horizontal line, respectively) are 1_a-equivalent (or 1_b-equivalent). When dealing with SV relations $\mathsf{s}_a \neq 1_a$ this correspondence of formula and diagram would not hold any more.

We define the *Pareto dominance region* $\mathsf{pd}(a, b, r) :: T_a \bowtie T_b$ by

$$\mathsf{pd}(a, b, r) =_{df} |c + 1_{a \bowtie b}\rangle(c \rhd r) \quad \text{with} \quad c = a \otimes b \ .$$

This is the inverse image of the maxima $(a \otimes b) \rhd r$ within $1_a \bowtie 1_b$ w.r.t. the reflexive closure of $a \otimes b$. For sake of readability we introduce the following abbreviation for any relation $a :: T^2$:

$$\underline{a} =_{df} a + 1_a \ .$$

The dominance region covers the grey area in Figure 4.2 and can be simplified by the following calculation for the argument of the diamond:

$$\begin{array}{ll}
& (a \otimes b) + (1_a \bowtie 1_b) \\
= & \quad \{\![\text{ definition, assumption of } \mathsf{s}_a = 1_a, \mathsf{s}_b = 1_b \ ]\!\} \\
& (1_a + a) \bowtie b \ + \ a \bowtie (1_b + b) \ + \ 1_a \bowtie 1_b \\
= & \quad \{\![\text{ distributivity of + over } \bowtie \ ]\!\} \\
& 1_a \bowtie b \ + \ a \bowtie b \ + \ a \bowtie 1_b \ + \ a \bowtie b \ + \ 1_a \bowtie 1_b \\
= & \quad \{\![\text{ idempotency of + }]\!\} \\
& 1_a \bowtie b \ + \ 1_a \bowtie 1_b \ + \ a \bowtie b \ + \ a \bowtie 1_b \\
= & \quad \{\![\text{ distributivity of + over } \bowtie \ ]\!\} \\
& 1_a \bowtie (1_b + b) \ + \ a \bowtie (1_b + b) \\
= & \quad \{\![\text{ again distributivity }]\!\} \\
& (1_a + a) \bowtie (1_b + b) \\
= & \quad \{\![\text{ convention for } \underline{a} \text{ and } \underline{b} \ ]\!\} \\
& \underline{a} \bowtie \underline{b} \ .
\end{array}$$

Hence we get

$$\mathsf{pd}(a, b, r) = |\underline{a} \bowtie \underline{b}\rangle((a \otimes b) \rhd r) \ .$$

Assume a maxima set of N tuples $x^{(1)}, ..., x^{(N)}$, i.e.,

$$(a \otimes b) \rhd r = x^{(1)} + ... + x^{(N)} \ ,$$

where $x^{(i)} = x_a^{(i)} \bowtie x_b^{(i)}$ with some $x_a^{(i)} :: T_a, x_b^{(i)} :: T_b$ holds by convention for all $i \in \{1, ..., n\}$. Using distributivity of diamond over +, the Pareto dominance region, shown in Figure 4.2, is given by

$$\mathsf{pd}(a, b, r) = q^{(1)} + ... + q^{(N)} \quad \text{with} \quad q^{(i)} =_{df} |\underline{a} \bowtie \underline{b}\rangle x^{(i)} \ .$$

Since the $x^{(i)}$ are rectangles, the $q^{(i)}$ are also rectangles because of

$$\begin{array}{rl}
q^{(i)} & = |\underline{a} \bowtie \underline{b}\rangle x^{(i)} \\
& = |\underline{a} \bowtie \underline{b}\rangle(x_a^{(i)} \bowtie x_b^{(i)}) \\
& = |\underline{a}\rangle x_a^{(i)} \bowtie |\underline{b}\rangle x_b^{(i)} \ .
\end{array}$$

Hence $\mathsf{pd}(a,b,r)$ can be represented as a sum of N rectangles. The complement of the dominance region can be written as the sum of $(N+1)$ rectangles. We define the complement of the Pareto dominance region by

$$\overline{\mathsf{pd}}(a,b,r) =_{df} \neg\mathsf{pd}(a,b,r) = 1_a \bowtie 1_b \; - \; \mathsf{pd}(a,b,r) \; .$$

Note that "$\bowtie$" binds stronger than "$-$" and "$+$".

In the following we will infer a more compact representation of $\overline{\mathsf{pd}}(a,b,r)$. At first we state a first lemma giving an indexation of the Pareto front.

Lemma 4.2.2. *Let $a :: T_a^2, b :: T_b^2$ be layered preferences. For the maxima set $(a \otimes b) \rhd r = x^{(1)} + ... + x^{(N)}$, the $x^{(i)}$ can always be arranged such that for all $i \in \{1,...,N-1\}$:*

a) $x_a^{(i)} \; \underline{a} \; x_a^{(i+1)}$

b) $x_b^{(i+1)} \; \underline{b} \; x_b^{(i)}$

c) $x^{(i)} \leq |\underline{a}\rangle x^{(i+1)}$

d) $x^{(i+1)} \leq |\underline{b}\rangle x^{(i)}$

Proof. As a is a layered preference, the arrangement (a) is obviously possible. Next, we show that this implies (b). Since b is layered we have one of the following cases:

i) $x_b^{(i)} \; b \; x_b^{(i+1)}$,

ii) $x_b^{(i+1)} \; b \; x_b^{(i)}$,

iii) $x_b^{(i)} \; 1_b \; x_b^{(i+1)}$.

Suppose case (i), i.e., $x_b^{(i)} \; b \; x_b^{(i+1)}$. Then $x^{(i)} \; (\underline{a} \bowtie b) \; x^{(i+1)}$ and hence $x^{(i)} \; (a \otimes b) \; x^{(i+1)}$, i.e., $x^{(i)}$ is dominated by $x^{(i+1)}$, a contradiction. Hence only the cases (ii) and (iii) are possible, which imply condition (b).

Parts (c) and (d) immediately follow from (a) and (b). □

The subsequent lemma gives a compact representation of $\overline{\mathsf{pd}}(a,b,r)$, i.e., a formal description of the white area in Figure 4.2.

Lemma 4.2.3. *The set $\overline{\mathsf{pd}}(a,b,r)$ can be expressed as a sum of $(N+1)$ rectangles as follows (where $\neg$ binds stronger than $\bowtie$):*

$$\overline{\mathsf{pd}}(a,b,r) = \neg q_a^{(1)} \bowtie 1_b \; + \; \neg q_a^{(2)} \bowtie \neg q_b^{(1)} \; + \; ... \; + \; 1_a \bowtie \neg q_b^{(N)} \; . \tag{4.1}$$

Proof. For convenience we set

$$q_a^{(0)} =_{df} 1_a, \;\; q_a^{(N+1)} =_{df} 0_a, \quad q_b^{(0)} =_{df} 0_b, \;\; q_b^{(N+1)} =_{df} 1_b \; .$$

From Lemma 4.2.2 we conclude for $i \in \{1,...,N-1\}$:

$$q_a^{(i+1)} = |a + 1_a\rangle x_a^{(i+1)} \leq |a + 1_a\rangle x_a^{(i)} = q_a^{(i)} \; .$$

Analogously we get $q_b^{(i)} \leq q_b^{(i+1)}$. With the above conventions $(q_a^{(i)})_{i=0,...,N+1}$ is decreasing while $(q_b^{(i)})_{i=0,...,N+1}$ is increasing in i. Hence the claim in Equation (4.1) is equivalent to

$$\overline{\mathsf{pd}}(a,b,r) = \sum_{i=0}^{N} \neg q_a^{(i)} \bowtie \neg q_b^{(i+1)} \; .$$

To show this, we check the usual test properties, i.e., $p + \neg p = 1_p$ and $p \cdot \neg p = 0_p$. In the concrete case we will show $\mathsf{pd}(a,b,r) \cdot \overline{\mathsf{pd}}(a,b,r) = 0_r$ and $\mathsf{pd}(a,b,r) + \overline{\mathsf{pd}}(a,b,r) = 1_a \bowtie 1_b$.

1. Remember that $\mathsf{pd}(a,b,r) = \sum_{i=1}^{N} q^{(i)}$. We have to show that all summands in $\mathsf{pd}(a,b,r) \cdot \overline{\mathsf{pd}}(a,b,r)$ are 0_r. We conclude for all i,j:

$$q^{(i)} \cdot (\neg q_a^{(j)} \bowtie \neg q_a^{(j+1)}) = \underbrace{(q_a^{(i)} - q_a^{(j)})}_{=:p_a} \bowtie \underbrace{(q_b^{(i)} - q_b^{(j+1)})}_{=:p_b} .$$

   Now we have either $j \leq i-1$ which implies that $p_a = 0_a$, or we have $j \geq i$ which implies that $p_b = 0_b$. Hence all summands are 0_r.

2. We use the following decomposition of $1_a \bowtie 1_b$:

$$1_a \bowtie 1_b = \sum_{i,j=0}^{N} \underbrace{(q_a^{(i)} - q_a^{(i+1)}) \bowtie (q_b^{(j+1)} - q_b^{(j)})}_{=:p_{i,j}} .$$

   Next, we show that any $p_{i,j}$ is contained either in $\mathsf{pd}(a,b,r)$ or in $\overline{\mathsf{pd}}(a,b,r)$. We distinguish two cases:

   (i) $j \leq i-1$: Because $q_b^{(i)}$ is increasing in i we have

$$p_{i,j} \leq q_a^{(i)} \bowtie q_b^{(j+1)} \leq q_a^{(i)} - q_b^{(i)} \leq \mathsf{pd}(a,b,r) .$$

   (ii) $j \geq i$: Because $\neg q_b^{(i)}$ is decreasing in i we have

$$p_{i,j} \leq \neg q_a^{(i+1)} \bowtie \neg q_b^{(j)} \leq \neg q_a^{(i+1)} \bowtie \neg q_b^{(i)} \leq \overline{\mathsf{pd}}(a,b,r) .$$

   Thus the sum of $\mathsf{pd}(a,b,r)$ and $\overline{\mathsf{pd}}(a,b,r)$ is equivalent to the entire domain $1_a \bowtie 1_b$ and hence the claim follows. □

Due to the above lemma we have a representation of $\overline{\mathsf{pd}}(a,b,p)$ with $(N+1)$ rectangles. To see this more concretely we show the special case of $N = 2$ in the following example.

Example 4.2.4. Let $\mathsf{pd}(a,b,r) = q^{(1)} + q^{(2)}$ be the dominance region. According to Lemma 4.2.3 we get for $\overline{\mathsf{pd}}(a,b,r)$:

$$\overline{\mathsf{pd}}(a,b,r) = \neg q_a^{(1)} \bowtie 1_b \;+\; \neg q_a^{(2)} \bowtie \neg q_b^{(1)} \;+\; 1_a \bowtie \neg q_b^{(2)} .$$

4.2.3 Application

In the following we sketch an application for the calculations shown above. Assume a data set $r :: T_a \bowtie T_b$ where new tuples $x = x_a \bowtie x_b :: T_a \bowtie T_b$ are successively added. For a given preference $a \otimes b$ we want to keep the Pareto front updated at any time. In the following we consider the update process $r' = r + x$ for a single tuple x. Hence the set $p = (a \otimes b) \rhd r$ has to be updated to $p' = (a \otimes b) \rhd (r + x)$. There are two possibilities: either the new tuple belongs to the Pareto front or it does not, i.e., $p' = p$ or $p' = p + x$. We sketch how the above calculations will help to determine which one is the case.

Assume a measure function $\mu : 1_r \to [0,1]$ representing the probability in which area of $1_r = 1_a \bowtie 1_b$ new tuples will occur. An algorithm for deciding quickly if a new tuple is within $\mathsf{pd}(a,b,r)$ or within $\overline{\mathsf{pd}}(a,b,r)$ should check the most probable rectangles w.r.t. μ first, i.e., we calculate $\mu(q^{(i)})$ for $i = 1,\ldots,N$ and $\mu(\neg q_a^{(i)} \bowtie \neg q_b^{(i+1)})$ for $i = 0,\ldots,N$. Then for a new tuple $x = x_a \bowtie x_b$ we check if $x \leq r^{(i)}$, where the sequence $(r^{(i)})$ enumerates

the $2N+1$ rectangles of $\mathsf{pd}(a,b,r)$ and $\overline{\mathsf{pd}}(a,b,r)$ in a μ-decreasing order. The algorithm terminates if $x \leq r^{(i)}$ is true which gives evidence whether x is in $\mathsf{pd}(a,b,r)$ or $\overline{\mathsf{pd}}(a,b,r)$.

If a new tuple x is in $\mathsf{pd}(a,b,r)$, then it is already dominated by the current maximal tuples, i.e., we have $p = p'$ and the new tuple can be safely ignored. If $x \leq \overline{\mathsf{pd}}(a,b,r)$ then we have $p' = p + x$ and the rectangles representing $\mathsf{pd}(a,b,r')$ and $\overline{\mathsf{pd}}(a,b,r')$ with $r' = r + x$ have to be recalculated. Note that it suffices to recalculate only those rectangles $q^{(i)}$ where the index i belongs to the set

$$I = \{i \in \{1,...,N\} \mid x^{(i)} \leq |a \otimes b\rangle x\} \ ,$$

i.e., those rectangles are affected, where the corresponding tuples $x^{(i)}$ are dominated by x. For example, if $I = \{k\}$, then the three rectangles

$$q^{(k)}, \quad \neg q_a^{(k-1)} \bowtie \neg q_b^{(k)}, \quad \neg q_a^{(k)} \bowtie \neg q_b^{(k+1)}$$

have to be recalculated. Note that by transitivity of a and b and the definition of the Pareto preference, the set I is always an interval in $\mathbb{N}$, i.e., $I = \{l_1, l_1+1, ..., l_2\}$ for $l_1, l_2 \in \mathbb{N}$. If we have $|I| = k$, then $2k+1$ rectangles will be replaced by three new rectangles.

For real-world applications one might not have the probability measure μ available, but it can be roughly estimated by the received tuples from the stream which are already known at a given time.

An algorithm for quickly deciding if a new tuple is dominated by the existing maxima set is of interest for the *Online Skyline problem*, where the Skyline computation is done in an interactive environment. In this context, the data set is typically very large and can be changed by the user while the algorithm is running. This problem is studied in [KRR02], where the Nearest Neighbour search contains a similar idea to our rectangle approach.

4.3 A Prefilter Example: The Scalagon Algorithm

The name *Scalagon* stems from the words *scaling* and Hexagon [PK07]. The idea of this Skyline algorithm is to first discretize the data set (with a fixed scaling function) and apply a discrete Skyline algorithm, exploiting the lattice properties of the Pareto preference. This is a prefilter step according to the notion of prefilters given in Section 3.4. In the next step, the maximal tuples within the prefiltered set are calculated with a usual comparison based algorithm like BNL. Examples for lattice algorithms, optimized for low cardinality discrete domains, are Hexagon [PK07] and Lattice Skyline [MPJ07]. The Scalagon algorithm has been published in [ERK15] and is implemented in the rPref package [Roo15f].

Subsequently, we will give an informal description of the algorithm. Next to this, we will point out the formal connection to prefilters.

4.3.1 The Algorithm

The Scalagon algorithm consists of four phases. We will explain them using the visualization of this process in Figure 4.3. Assume a preference $\mathsf{low}(x) \otimes \mathsf{low}(y) :: T_x^2 \bowtie T_y^2$ with a numerical domain $D_{T_x \bowtie T_x} = [0,1) \times [0,1)$.

1. The scaling and discretization function is applied to every tuple of the data set. In most cases a continuous domain $D \subseteq \mathbb{R}^d$ is mapped to hyperquadratic discrete domain $\{1,2,...,k\}^d$, where k depends on the *scaling factor*. We will not go into the details of the scaling process but refer to the paper [ERK15]. Regarding Figure 4.3,

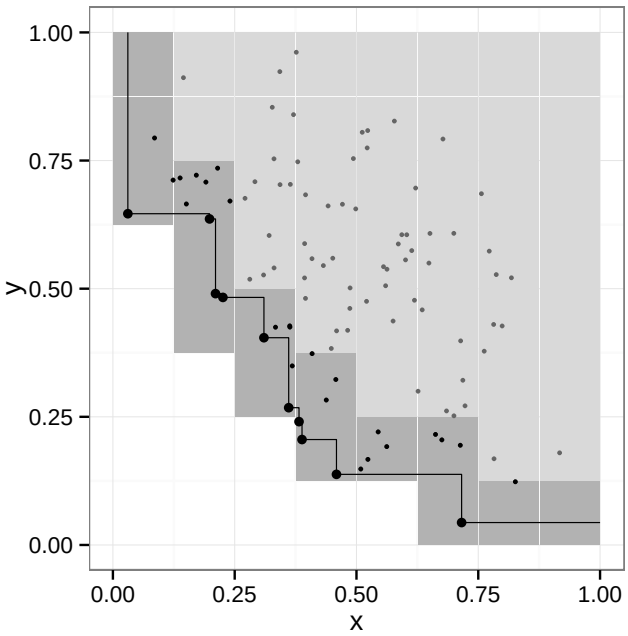


Figure 4.3: The prefiltering process of Scalagon: The light grey squares are dominated within the discretized data set. The dark grey squares contain the prefiltered set. Within this the final maxima, i.e., the Pareto front line, are calculated.

in this step the grid is established and each tuple is assigned to a tile. In this concrete case we have $d = 2$, $k = 8$ and the mapping is

$$f : [0,1)^2 \to \{1,...,8\}^2, (x,y) \mapsto (1 + \lfloor 8x \rfloor, 1 + \lfloor 8y \rfloor) \ .$$

2. Next, the maxima w.r.t. the product order on the discretized set are calculated. In the figure, this corresponds to determining the dark grey tiles.

3. In this phase all tuples belonging to the maximal tiles are picked for the next phase, i.e., all tuples within the dark grey tiles. With the calculation of this set, the prefiltering is done.

4. Finally a comparison based algorithm like BNL is used to determine the maxima within the prefiltered set. The result of this is the Pareto front, visualized by the bold points and the Pareto front line in Figure 4.3.

In the following section we present an algebraic argument for the correctness of this method. We will presume that we have correct algorithms for determining the maximal tiles on the discretized data set and a correct comparison based algorithm. A fine-grained algebraic proof of the well-known preference algorithm BNL, is shown in [Möl15].

4.3.2 Connection to Prefilters

Now we will describe the scaling and prefiltering process formally. First of all, let

$$a = a_1 \otimes ... \otimes a_d \quad \text{with} \quad a_i :: T_i^2$$

be a Pareto preference where all a_i are layered preferences. This is the input preference for the d-dimensional Skyline problem. Moreover we define the type $T = T_1 \bowtie ... \bowtie T_d$ such that $a :: T^2$ according to the definition of $\otimes$.

The scaling function is given by, where D_{T_i} are the type domains of type T_i,

$$f : D_{T_1} \times ... \times D_{T_d} \to S \quad \text{with} \quad S \subset \mathbb{N}^d \ ,$$
$$(x_1, ..., x_d) \mapsto (f_1(x_1), ..., f_d(x_d))$$

The function f is a valid scaling function if it is strictly increasing w.r.t. the orders $a_1, ..., a_d$ in the respective components f_i. Formally we require for all $i \in \{1, ..., k\}$

$$\forall x, y :: T_i : \quad f_i(x) < f_i(y) \ \Rightarrow \ x \, a_i \, y \ . \tag{4.2}$$

Note that the other direction ("$\Leftarrow$") is wrong in general, as f is a discretization.

In [ERK15] the domain only consists of real numbers, i.e., $D_{T_i} = \mathbb{R}$ for all i. The function f is an affine linear transformation followed by a ceiling operation. Thus we can write the components of f by $f_i(x) = \lceil \beta_i \cdot x + \gamma_i \rceil$. Further, all preferences a_i are given by $a_i = \mathsf{low}(T_i)$. It is clear that Equation (4.2) is fulfilled for such an f.

We now define a preference $b :: T^2$, which is the induced prefilter of this scaling, by

$$\forall x, y :: T : \quad x \, b \, y \ \Leftrightarrow \ f(x) < f(y) \ .$$

Together with Equation (4.2) this implies $b \leq a$. By Lemma 3.3.2.5 we get $a \rhd r \leq b \rhd r$ for all $r :: T$. According to Definition 3.4.1 we have $b \, \mathsf{pref}_r \, a$ for all $r :: T$. Hence picking tuples from the maximal tiles is a prefilter operation.

4.3.3 Performance Gain

To show that the prefilter concept leads indeed to a performance gain, we show the results of a performance study of Scalagon from [ERK15]. Thereby we are varying an "α-factor", determining the scaling (i.e., the factors β_i and γ_i in the affine linear transformation given above), which depends on the number of tuples, on the number of dimensions, and on the number of different values per dimension. We will not go into the details of the scaling calculation here and refer to [ERK15], Section 3.5.

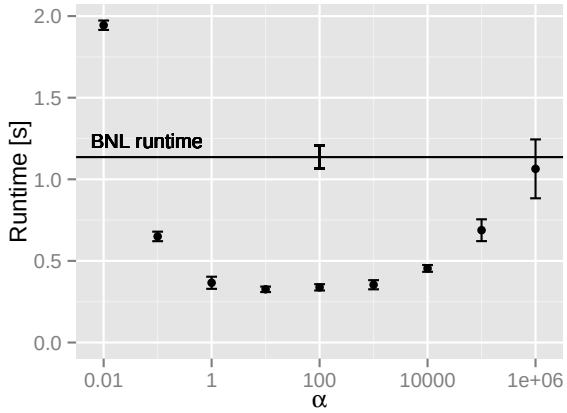


Figure 4.4: Runtime of Scalagon for different α values (logarithmic scale). For a wide area of α values, Scalagon is faster than BNL.

Roughly speaking, a small α value means that $|f(r)|$, i.e., the cardinality of the scaled data set, is large. For shifting $\alpha \to 0$ every tuple has its own tile at some point. A high α value means that $|f(r)|$ is small. For $\alpha \to \infty$ at some point there is just one tile, i.e., $|f(r)| = 1$. The first case is similar to the Hexagon algorithm [PK07], the second case corresponds to an empty prefilter. Note that it holds trivially by definition that $0_a \,\mathsf{pref}_r\, a$ is true. Hence the costs are the same as for the comparison based algorithms of phase 4, plus some computational overhead for the scaling.

Figure 4.4 shows a performance study of Scalagon, which is available as a script on the web [Roo15e]. For very small and very large values of α the runtime of a pure BNL approach is faster. But for a large range of values between, Scalagon is an efficient prefilter.

CHAPTER 5

Preference Decomposition

We study the expressiveness of preference query languages, i.e., the problem which preference operators and operands are necessary to construct different kinds of orderings. First, we introduce preference decomposition algorithms, transforming a given strict partial order into a term of logical preferences, the prioritisation and Pareto operator. Next, we prove their correctness based on the relational and algebraic approach. Finally, we introduce an optimized decomposition which especially improves the decomposition of power constructions of preferences, i.e., the lifting of a preference order to the power set of the domain. We will mostly consider preferences as concrete relations but we use the same notation and some of the algebraically derived results from the previous chapters. In this chapter we basically follow [Roo15c] and [Roo15a].

5.1 The Decomposition Problem

In the previous chapters we have considered given preference terms and investigated the induced strict partial order. Now we are considering a given strict partial order and try to find a preference term corresponding to that order. In Figure 5.1, an example of a Skyline and the Better-Than-Graph of the induced order is given.

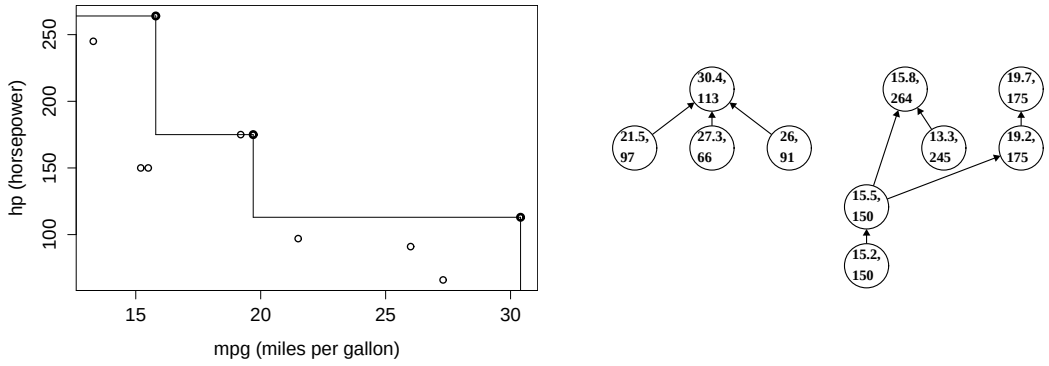


Figure 5.1: Pareto optima of cars (`mtcars[21:30,]` in R) with low fuel consumption (i.e., high miles per gallon *(mpg)* value) and high horsepower *(hp)*. The corresponding Better-Than-Graph containing the values for mpg and horsepower, is depicted alongside.

The task here is to characterize the *expressiveness* of a preference query language. We want to know which preference operators and operands are sufficient to construct different kinds of orderings. This hierarchy of orderings includes, among others, strict orders, weak orders and total orders.

In these studies, we are using the strong connection between the preference constructors defined in Chapter 2 and the query languages in Exasolution Skyline and the rPref package. Both implementations offer the $\{\&, \otimes\}$ operators and base preferences like is_true. Hence we can investigate the language expressiveness of these implementations by means of our calculus.

5.1.1 Motivation

In the preference framework from [Kie02] an *"Explicit(E)"-constructor* is suggested, which offers a direct solution for the decomposition problem. This constructor generates a preference from a given set of edges E, where the transitive hull of E is the *Better-Than-Graph*. This solves the decomposition problem in a very pragmatic way.

For example, let $r = x_1 + ... + x_4$ be a data set. We assume that the explicit-constructor expects a set of edges (x, y) where y is better than x. Then the preference

$$\text{explicit}(\{(x_3, x_2), (x_4, x_2), (x_2, x_1)\})$$

equals that from Figure 5.2(2) on page 61. Such generic constructors lack syntactic neatness and are error-prone, e.g., the user may accidentally fail to specify a strict order within E. For example, the term

$$\text{explicit}(\{(x_1, x_2), (x_2, x_3), (x_3, x_1)\})$$

would not result in a valid preference, as it contains a circle which implies a non-reflexive transitive hull. For those reasons, in the preference implementations Exasolution Skyline and rPref, such a constructor is not contained. In implementations containing this constructor like Preference SQL [KEW11] this problem is addressed by a cycle check before the evaluation, where a potential error is returned to the user.

Instead, we admit the fundamental preference operators $\{\&, \otimes\}$ together with Boolean preferences in our approach. The latter are the simplest preference constructs, as they just define Better-Than-Relations between two disjoint sets.

In the following we search for decompositions of any strict partial order using just these simple constructs. The answer to this problem precisely characterizes the expressiveness of the considered preference query languages.

Subsequently, we collect the building blocks for the decomposition approach. For the sake of readability we will model the tuples of the given data set by a unique identifier, corresponding to an atomic type in our type calculus. In SQL dialects, uniqueness is ensured for a *primary key*, which is usually the first column of a data set having the name `id`. We will do all operations on a single typed domain corresponding to such a unique identifier. This simplifies the notation largely without loosing generality. As we just consider finite data sets (according to the database use case) one can always find a unique enumeration of all tuples.

5.1.2 Formal Prerequisites

Let A be an atomic type with a type domain D_A. We will consider finite data sets $r :: A$ modelling the tuples which are a-related by a preference $a :: A^2$ which we want to

decompose. Let $a, b :: A^2$ be preferences of the same type. Then the $\ltimes$ operation occurring in the definitions of $a \,\&\, b$ and $a \otimes b$ (cf. Definition 3.2.7) collapses to the meet $\sqcap$, cf. axiom 3.1.6.7 of the join algebra. This means that the $\{\&, \otimes\}$ operators specialize to

$$a \,\&\, b =_{df} a \;+\; \mathsf{s}_a \sqcap b \;,$$
$$a \otimes b =_{df} (a + \mathsf{s}_a) \sqcap b \;+\; a \sqcap (b + \mathsf{s}_b) \;.$$

To ease readability, we will omit the type domain in the following. We assume that $a, b, ... :: A^2$ holds for all preferences and $p, q, r, ... :: A$ for all tuples and sets of tuples.

As preferences are transitive relations, we will consider their *Hasse diagrams*, i.e., the transitive reduction, for visualizing them as graphs. For a transitive relation a this is given by

$$a_h =_{df} a \sqcap \overline{a^2} \;,$$

cf. [SS93], Prop. 3.2.5. Because the Hasse diagram is unique for a preference, it can be also used for algorithmically testing if two preferences are identical, cf. Appendix A.1.

The decomposition approach is based on Boolean preferences, as given in Definition 2.2.4. We specialize this concept to preferences on sets and tuples.

Definition 5.1.1 (set/tuple preference)**.** For a set of tuples $p \leq 1_A$ we define the *set preference*

$$\mathsf{t}(p) =_{df} \mathsf{is\_true}(\rho_p) \text{ where } \rho_p(s) \Leftrightarrow_{df} \{(s, s)\} \leq p \;.$$

If $|p| = 1$, i.e., p is a tuple, we also say that $\mathsf{is\_true}(p)$ is a *tuple preference*. □

By this definition we have for all $x, y \in r$ that $(x \,\mathsf{t}(p)\, y) \Leftrightarrow (x \notin p \wedge y \in p)$ holds, i.e., all tuples in p are better than those in $\neg p$ w.r.t. $\mathsf{t}(p)$. We can also express this preference in a point-free fashion by $\mathsf{t}(p) = \neg p \cdot \top_A \cdot p$. Along with our convention that $p, q, ...$ are sets of tuples and $x, y, ...$ are tuples, we use $\mathsf{t}(p)$ for general set preferences, whereas $\mathsf{t}(x)$ refers to a tuple preference.

In the database setting, such a set preference can be easily realized with a Boolean preference on a primary column. For example `PREFERRING id in (1,2)` is a preference term in the Exasolution dialect which expresses the wish for tuples having an `id` value of 1 or 2, where `id` is the primary key of the table.

In the following we will investigate *preference terms* where Boolean preferences are connected by the $\{\&, \otimes\}$ operators. As a special case we consider terms with *unique tuple preferences*, where each tuple from the data set may occur at most once in the preference term. These preference terms have the pleasant property that the number of operands is restricted by the number of tuples and hence comparisons w.r.t. them are very efficient.

Consider the following example, where the data set $r = x_1 + ... + x_n$ with distinct tuples x_i is given: The tuple preferences in $a = \mathsf{t}(x_1) \,\&\, (\mathsf{t}(x_2) \otimes \mathsf{t}(x_3))$ are unique, because each x_i for $i \in \{1, 2, 3\}$ occurs just once. Instead, the preferences in $a = \mathsf{t}(x_1) \,\&\, (\mathsf{t}(x_2) \otimes \mathsf{t}(x_1))$ are not unique because x_1 occurs twice. Subsequently, we define uniqueness formally.

Definition 5.1.2 (uniqueness)**.** For a data set $r = x_1 + ... + x_n$, operators $\mathsf{op} \subseteq \{\&, \otimes\}$ and a preference a we say that the $\mathsf{t}(x_i)$ are *unique* in a if and only if a is in the set $\mathsf{un}_{\mathsf{op}}(r)$ recursively defined by:

$$\mathsf{un}_{\mathsf{op}}(s) =_{df} \begin{cases} \{0_A\} & \text{if } s = 0_A \;, \\ \{b \star \mathsf{t}(x) \mid x \in s,\, b \in \mathsf{un}_{\mathsf{op}}(s - x),\, \star \in \mathsf{op}\} \cup & \\ \quad \{\mathsf{t}(x) \star b \mid x \in s,\, b \in \mathsf{un}_{\mathsf{op}}(s - x),\, \star \in \mathsf{op}\} \;\cup\; \mathsf{un}_{\mathsf{op}}(s - x) & \text{otherwise} \;. \end{cases}$$

We will also call this set *unique tuple preferences* (over op). □

This set contains the closure under all operators in op and all operands in the set $\{\mathsf{t}(x) \mid x \in r\}$, which may occur just once in each preference.

In the last step of the recursion $\mathsf{un}_{\mathsf{op}}(0_A) = 0_A$ is obtained, hence $0_A \in \mathsf{un}_{\mathsf{op}}(s)$ for all s. This means, that all tuple preferences occurring in the empty preference 0_A (which are none) are unique by convention. In all the recursion steps $\mathsf{un}_{\mathsf{op}}(s)$ with $s \neq 0_A$ we get $0_A \star \mathsf{t}(x) = \mathsf{t}(x) \star 0_A = \mathsf{t}(x)$ because of neutrality of 0_A w.r.t. $\star \in \{\&, \otimes\}$.

To compare preference relations we introduce a concept of pairwise equivalence of preferences w.r.t. a given data set.

Definition 5.1.3 (r-equivalence)**.** Let a, b be preferences and r a data set. We say that a and b are r-equivalent, if and only if $r \cdot a \cdot r = r \cdot b \cdot r$.

This is equivalent to $(x\,a\,y) \Leftrightarrow (x\,b\,y)$ for all tuples $x, y \in r$. More illustratively, a preference a is r-equivalent to b if the Hasse diagrams of a and b on r are the same. Note that r-equivalence of a and b does *not* imply that $a \star c = b \star c$ for $\star \in \{\&, \otimes\}$ holds in general. We will show a counterexample in Remark 5.2.2.

Corollary 5.1.4. *Let r be a finite data set (i.e., with finitely many tuples) and a, b layered preferences. The prioritisation $a \,\&\, b$ is r-equivalent to a layered preference.*

Proof. Consider layered preferences a and b with measure functions f_a and f_b (cf. Corollary 3.2.3). By definition, $c = r \cdot (a \,\&\, b) \cdot r$ and $(a \,\&\, b)$ are r-equivalent. We have to show that c is a layered preference. Consider the measure function $f_c(x) = f_a(x) \cdot (1 + \max\{f_b(y) \mid y \in r\}) + f_b(x)$, where the maximum of f_b over r obviously exists because r is finite. We calculate for $x, y \in r$

$$\begin{aligned} x\,c\,y &\Leftrightarrow x\,a\,y \vee (x\,\mathsf{s}_a\,y \wedge x\,b\,y) \\ &\Leftrightarrow f_a(x) < f_a(y) \vee (f_a(x) = f_a(y) \wedge f_b(x) < f_b(y)) \Leftrightarrow f_c(x) < f_c(y)\ . \end{aligned}$$

Thus c is a layered preference with measure function f_c. □

Chains of $\otimes$-composition like $a_1 \otimes \ldots \otimes a_n$ can be interpreted as "better for one i and {better or equal for all i}", formally

$$x\,(a_1 \otimes \ldots \otimes a_n)\,y \iff (\exists i : x\,a_i\,y) \wedge (\forall i : x\,(a_i + \mathsf{s}_{a_i})\,y)\ . \tag{5.1}$$

This interpretation plays an important role for the intuition and the arguments in the remainder of this chapter.

5.1.3 The Expressiveness Problem

Now we have considered the complex preference operators $\{\&, \otimes\}$ and some specializations of Boolean preferences. In decreasing generality, these are a) set preferences, b) tuple preferences and c) unique tuple preferences. We are now interested whether any preference (i.e., strict partial order) can be expressed by using these operators and operands. Additionally, we want to determine which is the minimal (or most restricted) set of preference operators and operands being sufficient to express arbitrary preferences.

Table 5.1 summarizes the problem. Our aim is to characterize the sets corresponding to each *?*-cell in the table. The question for each cell is, if it corresponds to the entire class of preferences, or, if it is proper subclass, e.g., layered preferences, total orders, etc.

Note that it is trivial to express any preference by set preferences and $\{+, \sqcap\}$ as preference operators, instead of $\{\&, \otimes\}$: For a data set r and $x, y \in r$ consider that $a_{x,y} = \mathsf{t}(\neg x) \sqcap \mathsf{t}(y)$

Table 5.1: The expressiveness problem is to characterize which preference orders can be composed from the given operators and operands.

	unique tuple pref.	tuple preferences	set preferences
&	*?*	*?*	*?*
⊗	*?*	*?*	*?*
{&, ⊗}	*?*	*?*	*?*

constructs a preference where $(z\, a_{x,y}\, z')$ holds if and only if $z = x \wedge z' = y$. Hence a preference a has an r-equivalent decomposition via the union

$$\sum_{x,y \in r \,\wedge\, (x\, a\, y)} a_{x,y} \ .$$

By using $\{+, \sqcap, (\cdot)^{-1}\}$ as operators, even tuple preferences suffice as operands, because $a_{x,y} = \mathsf{t}(x)^{-1} \sqcap \mathsf{t}(y)$ holds.

But in the scope of database preferences the operators $\{\&, \otimes\}$ have been established as the most common operators for good reasons. They have the pleasant property that they preserve strict partial orders (& preserves even strict weak orders, cf. Corollary 5.1.4). In contrast, the union, i.e. $a_1 + a_2$, violates transitivity in general and its transitive closure $(a_1 + a_2)^+$ does not preserve irreflexivity in general. For example $(a + a^{-1})^+$ is not irreflexive for a non-empty preference a. Hence from the application level it seems reasonable to restrict a preference language to operators like $\{\&, \otimes\}$, and especially to exclude the union operator.

In the following we will present a first idea how different preferences can be constructed from these building blocks. The intuitive mechanism behind a set preference is to pick some tuples from the data set and declare them to be better than the remainder. Using &-chains of tuple preference we can construct total orders and by ⊗-composition we can combine such chains. We illustrate this in the following example.

Example 5.1.5. Consider the data set $r = x_1 + x_2 + x_3 + x_4$ and the following terms of unique tuple preferences. They are visualized in Figure 5.2(1–3).

1. $a = \mathsf{t}(x_1)$
2. $b = \mathsf{t}(x_1)\ \&\ \mathsf{t}(x_2)$
3. $c = \mathsf{t}(x_1)\ \&\ (\mathsf{t}(x_2) \otimes \mathsf{t}(x_3))$

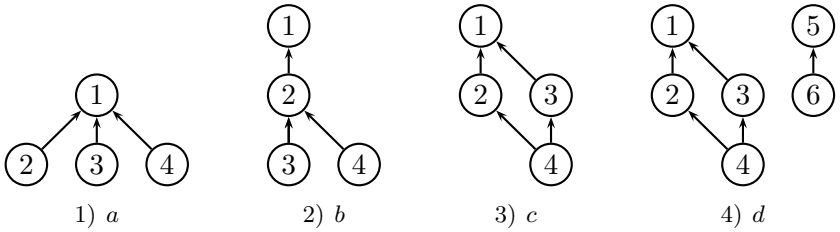


Figure 5.2: Hasse diagrams of preferences from Example 5.1.5 (a, b, c) and Remark 5.1.6 (d), where a circled i is short for x_i.

Comparing the preference terms of a, b, c and their Hasse diagrams in Figure 5.2, one recognizes that parallel compositions in the graph correspond to ⊗ and serial compositions to &. This unveils the following interesting link to language theory.

Remark 5.1.6. The paths in the Hasse diagram of a data set r w.r.t. a preference $d \in \mathsf{un}_{\mathsf{op}}(r)$ coincide with the language of a finite automaton without loops where every edge label occurs just once. At the language level, this implies that concatenation and choice are sufficient to describe the set of words, induced by the diagram of a preference within $\mathsf{un}_{\mathsf{op}}(r)$. For example, the preference

$$d = (\mathsf{t}(x_1) \,\&\, (\mathsf{t}(x_2) \otimes \mathsf{t}(x_3)) \,\&\, \mathsf{t}(x_4)) \otimes (\mathsf{t}(x_5) \,\&\, \mathsf{t}(x_6)),$$

on $r = x_1 + ... + x_6$ (see Figure 5.2(4)) corresponds to the behaviour (cf. [Eil74])

$$\{x_4\}\,(\{x_2\} \cup \{x_3\})\,\{x_1\} \;\cup\; \{x_6\}\{x_5\}\ .$$

There & corresponds to concatenation and $\otimes$ to choice.

Example 5.1.5 shows that some preferences can be expressed with $\{\&, \otimes\}$ and unique tuple preferences. This raises the question if this restricted set of operands already suffices to express arbitrary strict partial orders. The following lemma shows that this is not the case.

Lemma 5.1.7. *Let $r = x_1 + ... + x_n$ be a data set, where x_i are its distinct tuples. Then the following proper inclusions in the sense of r-equivalence hold:*

$$\{\text{ layered preferences }\} \;\subsetneq\; \mathsf{un}_{\{\&,\otimes\}}(r) \;\subsetneq\; \{\text{ preferences }\}$$

Proof. First we show the "$\subseteq$" conditions from left to right. Let a be a layered preference and f_a be the measure function of a according to Corollary 5.1.4. Straightforward verification shows that a is r-equivalent to a' where

$$a' = \mathop{\&}_{i=0}^{N-1} (\mathsf{t}(y_{i,1}) \otimes ... \otimes \mathsf{t}(y_{i,n_i})) \quad \text{with} \quad \{y_{(N-i),1}, ..., y_{(N-i),n_i}\} := f_a^{-1}(i),$$

for $i = 1, ..., N$ with $N := \max\{f_a(x) \mid x \in r\}$.

The preimages $f_a^{-1}(i)$ are disjoint for distinct i, hence the $\mathsf{t}(y_{i,j})$ are unique in a'. This implies $a' \in \mathsf{un}_{\{\&,\otimes\}}(r)$ which shows the first inclusion. The next "$\subseteq$" inclusion is trivially true, as preferences are closed under $\{\&, \otimes\}$.

Regarding the proper subset conditions consider the following preferences (visualized in Figure 5.3(1–2)) that falsify equality.

1. Let $r = x_1 + x_2 + x_3$ and $a = (\mathsf{t}(x_1) \,\&\, \mathsf{t}(x_2)) \otimes \mathsf{t}(x_3)$. Obviously we have $a \in \mathsf{un}_{\{\&,\otimes\}}(r)$, but a is not a layered preference because negative transitivity ($(\overline{a})^2 = \overline{a}$) is violated:

$$(x_2\,\overline{a}\,x_3) \wedge (x_3\,\overline{a}\,x_1) \quad (\text{implying } x_2\,(\overline{a})^2\,x_1) \qquad \text{but} \quad \neg(x_2\,\overline{a}\,x_1)\ .$$

2. Let $r = x_1 + ... + x_4$ and $b = ((\mathsf{t}(x_1) \otimes \mathsf{t}(x_3)) \,\&\, \mathsf{t}(x_2)) \otimes (\mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4))$. We show that there is no r-equivalent preference in $\mathsf{un}_{\{\&,\otimes\}}(r)$ by a model finder checking all possible preference terms, see Appendix A.1. □

Remark 5.1.8. In Remark 5.1.6 we already noticed a connection between the Hasse diagrams of preferences in $\mathsf{un}_{\{\&,\otimes\}}(r)$ and concatenation/choice in formal languages. Their induced orders coincide with series-parallel pomsets as described in [Gis88]. Finite pomsets are series-parallel if and only if they are *N-free* (Theorem 3.1 in [Gis88]), where N-free means that the diagram contains no "N-shaped" subrelation like the preference b ($b \notin \mathsf{un}_{\{\&,\otimes\}}(r)$, cf. Figure 5.3(2)) from the above proof.

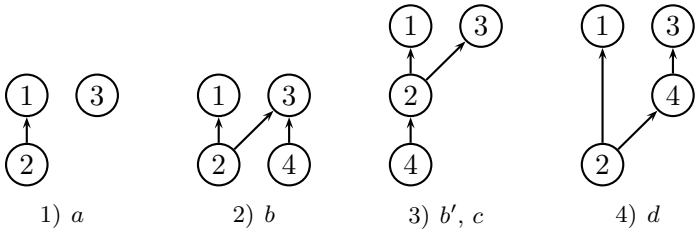


Figure 5.3: Hasse diagrams of preferences for Lemma 5.1.7, where a circled i is short for x_i.

To get an intuitive understanding why this preference b cannot be decomposed within $\mathsf{un}_{\{\&,\otimes\}}(r)$ consider the layered preference $b' = (\mathsf{t}(x_1) \otimes \mathsf{t}(x_3)) \,\&\, \mathsf{t}(x_2)$. Inserting "$\&\,\mathsf{t}(x_4)$" into b' in such a way that $(x_4\, a\, x_3)$ holds is possible at two positions (the resulting preferences are visualized in Figure 5.3(3–4)):

1. We insert "$\&\,\mathsf{t}(x_4)$" at the end of b', i.e., we get the preference

$$c = b' \,\&\, \mathsf{t}(x_4) = (\mathsf{t}(x_1) \otimes \mathsf{t}(x_3)) \,\&\, \mathsf{t}(x_2) \,\&\, \mathsf{t}(x_4) \ .$$

   Then b' and c are r-equivalent and $(x_4\, c\, x_2)$ holds, which we do not want.

2. We replace $\mathsf{t}(x_3)$ by $(\mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4))$, i.e., we get

$$d = (\mathsf{t}(x_1) \otimes (\mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4))) \,\&\, \mathsf{t}(x_2)) \ .$$

   This causes the undesired effect of $(x_2\, d\, x_4)$.

In both cases, the tuples x_2 and x_4 tuples are in a better-than-relation and hence the desired incomparability of these tuples cannot be obtained.

Now we omit the restriction to tuple preferences and consider general set preferences. First, we simplify the construction from Lemma 5.1.7 for the special case of layered preferences by using set preferences. The idea is to express a layer $p = x_{i_1} + ... + x_{i_n}$ within a prioritisation chain simply by $\mathsf{t}(p)$ instead of $\mathsf{t}(x_{i_1}) \otimes ... \otimes \mathsf{t}(t_{i_n})$. We clarify this in the subsequent lemma.

Lemma 5.1.9. *Let r be a data set. In the sense of r-equivalence the set of layered preferences over r is equivalent to &-chains of set preferences over r, formally*

$$\{ \text{ layered preferences } \} = \{\mathsf{t}(p_1) \,\&\, \mathsf{t}(p_2) \,\&\, ... \,\&\, \mathsf{t}(p_k) \mid p_1 + ... + p_k \leq r\} \ .$$

Proof. Let a be a layered preference on r. A straightforward verification shows that the following preference b is r-equivalent to a:

$$b = \mathop{\&}_{i=0}^{N-1} \mathsf{t}(y_{i,1} + ... + y_{i,n_i}) \quad \text{where} \quad \{y_{(N-i),1}, ..., y_{(N-i),n_i}\} := f_a^{-1}(i),$$

for $i = 1, ..., N$ with $N := \max\{f_a(x) \mid x \in r\}$.

The reverse inclusion is clear, as set preferences are layered preferences and this property is preserved by &, cf. Corollary 5.1.4. □

An application of this lemma is a compact representation of the induced layered preference from Definition 3.5.6, which we can now rewrite via

$$m(a, r) =_{df} \mathop{\&}_{i=1}^{N} \mathsf{t}(q_i) \ ,$$

where the q_i and N are given as in Definition 3.5.3, and Lemma 3.5.5.3, respectively.

5.1.4 Decompositions of General Preferences

The results up to now show that unique tuple preferences do not suffice to construct arbitrary preferences. Subsequently, we will search for decompositions under less restrictive conditions. Primarily, we will investigate the following cases:

i) (Non-unique) tuple preferences and $\{\&, \otimes\}$,

ii) set preferences and only $\otimes$.

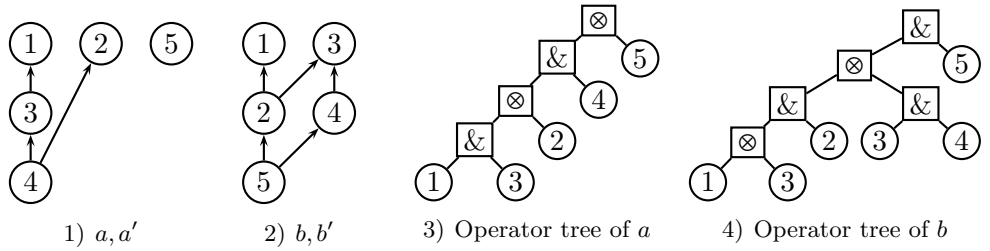


Figure 5.4: Hasse diagrams and operator trees of preferences from Example 5.1.10. In the operator trees a circled i is short for $\mathsf{t}(x_i)$.

Example 5.1.10. The preferences depicted in Figure 5.4 on the data set $r = \sum_{i=1}^{5} x_i$ can be decomposed in the following ways:

1. Using unique tuple preferences we find the following decomposition of the preference, depicted in Figure 5.4(1):

$$a = (((\mathsf{t}(x_1) \,\&\, \mathsf{t}(x_3)) \otimes \mathsf{t}(x_2)) \,\&\, \mathsf{t}(x_4)) \otimes \mathsf{t}(x_5) \ .$$

   Alternatively we can decompose the preference a into a $\otimes$-composition of set preferences, where each set is closed under $\langle a + 1_A |(\cdot)$, i.e., upward closed:

$$a' = \mathsf{t}(x_1) \otimes \mathsf{t}(x_2) \otimes \mathsf{t}(x_1 + x_3) \otimes \mathsf{t}(x_1 + x_2 + x_3 + x_4) \otimes \mathsf{t}(x_5) \ .$$

2. For the preference depicted in Figure 5.4(2) we get analogously

$$b = (((\mathsf{t}(x_1) \otimes \mathsf{t}(x_3)) \,\&\, \mathsf{t}(x_2)) \otimes (\mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4))) \,\&\, \mathsf{t}(x_5) \ ,$$
$$b' = \mathsf{t}(x_1) \otimes \mathsf{t}(x_3) \otimes \mathsf{t}(x_1 + x_2 + x_3) \otimes \mathsf{t}(x_3 + x_4) \otimes \mathsf{t}(r) \ .$$

The example above leads us to the conjecture that $\otimes$-compositions of set preferences *or* tuple preferences with $\{\&, \otimes\}$ are sufficient to express arbitrary preferences up to r-equivalence. The construction principle seems to be strongly connected to the Hasse diagram of the preference. Roughly speaking, we use $\otimes$ to connect parallel chains and $\&$ to step down in the Hasse diagram, cf. the operator trees in Figure 5.4(3–4). For the construction of $\otimes$-chains of set preferences, every element from the data set is transformed to a set containing all its successors w.r.t. the preference. In the next section we will formalize these constructions and give correctness proofs.
Our results are summarized in Table 5.2 and its rows are subsequently justified:

1. For $\&$-connected tuple preferences (first two cells) consider that a total order

$$(x_1 \, a \, x_2) \wedge (x_2 \, a \, x_3) \wedge \ldots \wedge (x_{n-1} \, a \, x_n)$$

Table 5.2: Expressiveness of different preference operators/operands. All inclusions and equalities are in the sense of r-equivalence.

	unique tuple pref.	tuple preferences	set preferences
$\&$	$\supsetneq$ total orders $\subsetneq$ layered pref.	$\supsetneq$ total orders $\subsetneq$ layered pref.	layered preferences *(Lemma 5.1.9)*
$\otimes$	set preferences	set preferences	*preferences?*
$\{\&, \otimes\}$	$\supsetneq$ layered pref. $\subsetneq$ preferences *(Lemma 5.1.7)*	*preferences?*	*preferences?*

can be expressed as the preference

$$a = \mathtt{t}(x_n) \,\&\, \mathtt{t}(x_{n-1}) \,\&\, ... \,\&\, \mathtt{t}(x_1) \ .$$

Not all &-chains of tuple preferences are total orders. Consider the case where at least two tuples x from the data set have no corresponding $\mathtt{t}(x)$ preference in the term. For example, the simple preference $\mathtt{t}(x_1)$ in Figure 5.2(1) is not a total order.

By Lemma 5.1.9, &-chains of set preferences are layered preferences, justifying the last cell of the first line. As tuple preferences are a special case of set preferences, the proper subset conditions "$\subsetneq$ layered pref." in the first two cells are also justified by Lemma 5.1.9. To see that &-connected tuple preferences are a proper subset of layered preferences, consider $\mathtt{t}(x_1 + x_2) \,\&\, \mathtt{t}(x_3 + x_4)$. This preference cannot be expressed with &-chains of tuple preferences.

2. All $\otimes$-compositions of (unique) tuple preferences can be expressed by set preferences, as $\mathtt{t}(x_1) \otimes ... \otimes \mathtt{t}(x_n)$ is r-equivalent to $\mathtt{t}(x_1 + ... + x_n)$ for tuples $x_i \in r$. This justifies the first two cells of the second line, while the last cell corresponds to our conjecture above.

3. The first cell of the last line is the result of Lemma 5.1.7 while the next two cells are again consequences of our conjecture, which will be proved later.

The results summarized in Table 5.2 show that $\otimes$-compositions of set preferences *or* $\{\&, \otimes\}$-compositions of tuple preferences are the simplest hypothetical possibility to decompose general preferences within this operators/operands matrix. We have shown that the other cells contain proper subsets of preferences.

In the following section we present two decomposition algorithms for preferences. By showing their correctness we will prove the conjecture related to the *"preferences?"* cells in the table.

5.2 Decomposition Algorithms

First, we will show the decomposition into set preferences and $\otimes$, followed by the decomposition algorithm for tuple preferences and $\{\&, \otimes\}$. The first one is a simple one-liner, the second algorithm needs some temporary variables, mainly some annotations of the Hasse diagram of the preference. The correctness proofs are quite technical and mostly in a point-wise fashion. Even though some arguments within the proofs are shown point-free and are based on the results from Chapter 2 and 3, we did not find an approach to show the correctness completely point-free.

5.2.1 Pareto Compositions of Set Preferences

The following theorem formalizes the decomposition of arbitrary preferences into ⊗-compositions of set preferences.

Theorem 5.2.1. *Let a be a preference and $r \leq 1_A$ a finite data set. Then a can be decomposed into a ⊗-composition of set preferences where each set is upward closed w.r.t. $a + 1_A$. Formally,*

$$\textsc{Decomp\_Pareto}(a, r) =_{df} \bigotimes_{x \in r} \mathsf{t}(r \cdot \langle a + 1_A | x) \ ,$$

is r-equivalent to a.

Proof. Let $b = \textsc{Decomp\_Pareto}(a, r)$ and $r = \sum_{i=1}^{k} x_i$ with distinct tuples x_i. We define the family of sets

$$p_i = r \cdot \langle a + 1_A | x_i \quad \text{for } i = 1, ..., k \ ,$$

corresponding to the arguments of the $\mathsf{t}(\cdot)$ preferences (i.e., preferred sets) contained in b. For this family it clearly holds that $\sum_{i=1}^{k} p_i = r$. The claim is $u\, a\, v \Leftrightarrow u\, b\, v$ for all $u, v \in r$. We show both implications separately:

"⇒" Assume $u, v \in r$ with $(u\, a\, v)$. We show first that $u \leq p_i \Rightarrow v \leq p_i$ holds for all $i \in \{1, ..., k\}$. From $u, v \in r$ with $(u\, a\, v)$ we get $v \leq r \cdot \langle a | u$. With this, the assumption $u \leq p_i$ and the transitivity of a we calculate:

$$v \leq r \cdot \langle a | u \leq r \cdot \langle a | (r \cdot \langle a + 1_A | x_i) \leq r \cdot \langle a^2 + a | x_i = r \cdot \langle a | x_i \leq r \cdot \langle a + 1_A | x_i = p_i \ .$$

The result above and the definition of $\mathsf{t}(\cdot)$ shows that u is not better than v w.r.t. all the $\mathsf{t}(\cdot)$ preferences in b, formally $\neg(v\, \mathsf{t}(p_i)\, u)$ for all i. As set preferences are layered preferences, this implies $(u\, (\mathsf{t}(p_i) + \mathsf{s}_{\mathsf{t}(p_i)})\, v)$. To finally prove that $(u\, b\, v)$ holds, according to the definition of ⊗ we have to show that there is at least one $\mathsf{t}(\cdot)$ for which v is better than u. For this we consider

$$u\, \mathsf{t}(r \cdot \langle a + 1_A | v)\, v \ ,$$

where $\mathsf{t}(r \cdot \langle a + 1_A | v)$ is by definition one of the ⊗-operands in b. This predicate is true because:

- It is clear that $v \leq r \cdot \langle a + 1_A | v$, i.e., v is in the preferred set.
- From $(u\, a\, v)$ it follows that $u \leq |a\rangle v$ and we have $(\langle a + 1_A | v) \cdot (|a\rangle v) = 0_A$ because a is a strict partial order. Hence $u \cdot r \cdot \langle a + 1_A | v = 0_A$, i.e., u is not in the preferred set.

Therefore, according to the definition of $\mathsf{t}(\cdot)$ we get that v is better than u w.r.t. $\mathsf{t}(r \cdot \langle a + 1_A | v)$ and not worse w.r.t. the other set preferences in b. By definition of ⊗ we conclude $(u\, b\, v)$.

"⇐" We show the contraposition $\neg(u\, a\, v) \Rightarrow \neg(u\, b\, v)$ for all atomic $u, v \in r$. We distinguish the following cases:

1. Assume $(v\, a\, u)$, i.e., u is better than v. Completely analogously to the first part of the proof we get $(v\, b\, u)$ and hence $\neg(u\, b\, v)$ as b is a strict partial order.

2. Otherwise we have $\neg(u\,(a + a^{-1})\,v)$, i.e., u and v are incomparable w.r.t. a. We can assume that $u \neq v$ holds because for $u = v$ the claim is trivially true as a is irreflexive. From the assumption we get that $u \cdot r \cdot \langle a + 1_A | v = 0_A$ and $v \cdot r \cdot \langle a + 1_A | u = 0_A$ holds. Hence we obtain

$$u\,\mathsf{t}(r \cdot \langle a + 1_A | v)\,v \;\wedge\; v\,\mathsf{t}(r \cdot \langle a + 1_A | u)\,u\ .$$

In the definition of b the preferences $\mathsf{t}(r \cdot \langle a + 1_A | x)$ are operands of a $\otimes$-composition for all $x \in r$. Especially there exists an operand with $x = u$ and one with $x = v$.

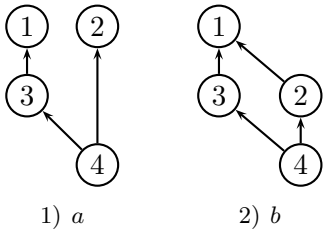
Hence this implies that u, v are incomparable w.r.t. b. Thus we finally get $\neg(u\,b\,v)$, which shows the claim. □

Remark 5.2.2. The preference generated according to Theorem 5.2.1 contains tuple preferences for each tuple from the maximal elements. Formally we have

$$\textsc{Decomp\_Pareto}(a, r) = \mathsf{t}(y_1) \otimes \ldots \otimes \mathsf{t}(y_n) \otimes \ldots \quad \text{with } a \rhd r = \sum_{i=1}^{n} y_i\ ,$$

where $y_i \in r$ are tuples. Even if $\mathsf{t}(y_1) \otimes \ldots \otimes \mathsf{t}(y_n)$ is r-equivalent to $\mathsf{t}(y_1 + \ldots + y_n)$ a substitution of $(\mathsf{t}(y_i) \otimes \mathsf{t}(y_j))$ by $\mathsf{t}(y_i + y_j)$ in the generated preference is not allowed in general. Consider the following counterexample, visualized in Figure 5.5. Let $r = x_1 + \ldots + x_4$. The following preferences a and b are not r-equivalent:

$$a = \mathsf{t}(x_1) \otimes \mathsf{t}(x_2) \otimes \mathsf{t}(x_1 + x_3) \otimes \mathsf{t}(x_1 + x_2 + x_3 + x_4)\ ,$$
$$b = \mathsf{t}(x_1 + x_2) \otimes \mathsf{t}(x_1 + x_3) \otimes \mathsf{t}(x_1 + x_2 + x_3 + x_4)\ .$$



1) a 2) b

Figure 5.5: Hasse diagrams of preferences for Remarks 5.2.2 and 5.2.5.

This means that we need the "induced incomparability" generated by the $\otimes$-connected tuple preferences. In the proof of Theorem 5.2.1 this is implicitly used in the last step, i.e., in the argument exploiting that the predicates $u\,\mathsf{t}(r \cdot |a + 1_A\rangle v)\,v$ and $v\,\mathsf{t}(r \cdot |a + 1_A\rangle u)\,u$ are simultaneously fulfilled. After a substitution of $\mathsf{t}(x) \otimes \mathsf{t}(y)$ by $\mathsf{t}(x + y)$ these predicates may not hold anymore. Hence, such a substitution invalidates the argument in the proof.

5.2.2 Pareto Compositions and Prioritisations of Tuple Preferences

Now we present an algorithm which decomposes any preference into tuple preferences with $\{\&, \otimes\}$ as composition operators. First we sketch the basic idea:

- Initially, the Hasse diagram and the maxima of the given data set r w.r.t. a preference a are calculated.

- Starting with the maxima as the *working set*, every node (i.e., tuple) $x \in r$ of the Hasse diagram will be annotated with a preference, expressing a partial decomposition of a which contains only those better-than-relations concerning the nodes above x in the Hasse diagram. Formally, the annotated preference is an $(r \cdot \langle a + 1_A | x)$-equivalent decomposition of a. To calculate this preference for each node $x \in r$ we

  1. take the annotations of all successors of x, and $\otimes$-compose them,
  2. add $\ldots \,\&\, \mathtt{t}(x)$ to this preference.

  This process is iterated downwards the Hasse diagram until the minima in r w.r.t. a are reached. This means that in each step the working set is replaced by the set of its maximal predecessors.

- Before the replacement, the annotations of all those nodes having predecessors are deleted (i.e., set to 0_A) after they were used to calculate the preferences of the predecessors. This step removes redundancy and leads to shorter preference terms. Formally, this step would not be necessary, as we will see in Lemma 5.2.3.

- Finally all non-zero annotations are $\otimes$-composed and returned. We claim that the returned preference is r-equivalent to a.

These annotations are stored in the preference-valued array $b[\cdot]$. The assignment $b[p] \leftarrow c$ for non-atomic and non-empty p is used as a shorthand notation for simultaneous assignments $b[x] \leftarrow c$ for all $x \in p$. We also assume that in all assignments the neutrality of 0_A is used, implying that $b[p] \leftarrow 0_A \star c$ is executed as $b[p] \leftarrow c$ for $\star \in \{\&, \otimes\}$.

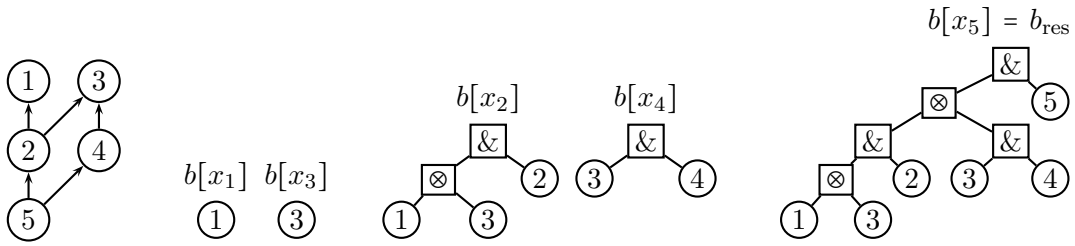
Algorithm 5.1 Preference decomposition into tuple preferences and $\{\&, \otimes\}$

Input: Preference to decompose a, data set r
Output: r-equivalent decomposition b_{res}

```
1:  function DECOMP_TUPLE(a, r)
2:      a_h ← a ⊓ overline(a^2)                          // Hasse diagram
3:      b[r] ← 0_A                                       // Initialization of array b of preferences
4:      m ← a ▷ r                                        // Start traversing with maxima
5:      while m ≠ 0_A do
6:          for all y ∈ m do                             // Pref. for y, collect and ⊗-compose successors,
7:              b[y] ← (⊗_{x∈r·⟨a_h|y} b[x]) & t(y)       // and finally add pref. on y
8:          end for
9:          b[r · ⟨a_h|m] ← 0_A                          // Delete preferences of m-successors
10:         m ← a ▷ (r · |a_h⟩m)                         // Find a-maximal predecessors of m
11:     end while
12:     b_res ← ⊗_{x∈r} b[x] ; return b_res              // ⊗-compose final preference and return
13: end function
```

In Figure 5.6 an example run of the algorithm is visualized where for every step the operator trees of the preferences in $b[\cdot]$ are shown.

Regarding the proof of correctness we will first get rid of line 9 of the algorithm, where the preferences of m-successors are deleted. This line removes redundant preferences, e.g., compare the final preference b_{res} from Figure 5.6 with b'_{res}, generated by the same



1) a 2) Iteration 1 3) Iteration 2 4) Iteration 3 and result

Figure 5.6: Example run of Algorithm 5.1 with $r = x_1 + \ldots + x_5$ and a, given by its Hasse diagram in (1). The $b[x]$ values in every iteration of the while loop are depicted as operator trees (2–4) where a circled i is short for $\mathsf{t}(x_i)$. We have $b[x] = 0_A$ for all values of $x \in r$ which are not depicted in each iteration.

algorithm, where line 9 is skipped:

$$\begin{aligned} b_{\text{res}} &= \quad (((\mathsf{t}(x_1) \otimes \mathsf{t}(x_3)) \,\&\, \mathsf{t}(x_2)) \otimes (\mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4))) \,\&\, \mathsf{t}(x_5) \\ b'_{\text{res}} &= b_{\text{res}} \otimes ((\mathsf{t}(x_1) \otimes \mathsf{t}(x_3)) \,\&\, \mathsf{t}(x_2)) \otimes (\mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4)) \otimes \mathsf{t}(x_1) \otimes \mathsf{t}(x_3) \end{aligned} \tag{5.2}$$

Hence line 9 ensures that the algorithm generates much simpler preference terms. Formally they are equivalent, as we state in the following lemma.

Lemma 5.2.3. *Let a, b preferences. Then $a \,\&\, b = a \otimes (a \,\&\, b)$ holds in the sense of strict equivalence (not just r-equivalence).*

Proof. With neutrality of 0_A for $\{\&, \otimes\}$ and the left-distributivity of & over $\otimes$ from Theorem 3.6.2, we obtain $a \,\&\, b = a \,\&\, (0_A \otimes b) = (a \,\&\, 0_A) \otimes (a \,\&\, b) = a \otimes (a \,\&\, b)$. □

If we apply this iteratively from right to left to b'_{res} in Equation (5.2) we immediately see that $b'_{\text{res}} = b_{\text{res}}$ holds. This implies that line 9 from Algorithm 5.1 can be removed without changing the result formally. We will use this in the proof of the following theorem, stating the correctness of Algorithm 5.1.

Theorem 5.2.4. *Let a be a preference and $r \leq 1_A$ a finite data set. The preference $b = \text{DECOMP\_TUPLE}(a, r)$ (from Algorithm 5.1) is r-equivalent to a.*

Proof. We reconsider the algorithm with a helper variable p summing up the traversed nodes of the Hasse diagram, and without the deletions in $b[\cdot]$, as explained above. We formulate a loop invariant (I) for the while loop stating

1) the working set m contains the a-maximal elements of the remainder $(r - p)$ and the traversed nodes p are contained in the data set r,

2) $b[y]$ is $(r \cdot \langle a + 1_A | y)$-equivalent to a (and also their SV relations are identical),

3) y and all tuples above y in the Hasse diagram are preferred w.r.t. $b[y]$ over the remainder $(\neg q)$ and $\neg q$ is one equivalence class w.r.t. $\mathsf{s}_{b[y]}$.

Formally we specify (I) ($\Leftrightarrow$ (I1) $\wedge$ (I2) $\wedge$ (I3)) by

$$\text{(I)} \Leftrightarrow_{df} \quad a \rhd (r - p) = m \;\wedge\; p \leq r \;\wedge \tag{I1}$$

$$\forall y \in p: \; (\, b[y] = q \cdot a \cdot q \;\wedge\; \mathsf{s}_{b[y]} = q \cdot \mathsf{s}_a \cdot q \;\wedge \tag{I2}$$

$$\mathsf{t}(q) \leq b[y] \;\wedge\; \neg q \cdot \mathsf{s}_{\mathsf{t}(q)} \cdot \neg q \leq \mathsf{s}_{b[y]} \;\text{ where }\; q = r \cdot \langle a + 1_A | y \,) \tag{I3}$$

Our proof strategy is to show that (I) is indeed an invariant and then conclude that b_{res} is r-equivalent to a. The modified algorithm reads as follows:

1: $a_h \leftarrow a \sqcap \overline{a^2}$; $b[r] \leftarrow 0_A$; $m \leftarrow a \rhd r$; $p \leftarrow 0_A$ // Initialization
2: {{ *(I) is true* }} // The invariant initially
3: **while** $m \neq 0_A$ **do**
4: $p \leftarrow p + m$ // Helper variable for the invariant
5: **for all** $y \in m$ **do**
6: $b[y] \leftarrow \left(\bigotimes_{x \in r \cdot \langle a_h | y} b[x]\right) \,\&\, \mathsf{t}(y)$
7: **end for**
8: $m \leftarrow a \rhd (r \cdot |a_h\rangle m)$
9: {{ *(I) is true* }} // The invariant after any loop run
10: **end while**
11: $b_{\text{res}} \leftarrow \bigotimes_{x \in r} b[x]$

First, we see that (I) trivially holds initially in line 2. Let (I)' ($\Leftrightarrow$ (I1)' $\wedge$ (I2)' $\wedge$ (I3)') be the invariant for the previous loop run, where $b'[\cdot]$ are the corresponding $b[\cdot]$-values. We have to show that (I)' $\Rightarrow$ (I) holds in line 9.

To see that (I1) holds, consider the p-update $p \leftarrow p+m$ in line 4. Using the strict partial order property of a we get that line 8 of the algorithm is equivalent to $m \leftarrow a \rhd (r-p)$, i.e., the working set is replaced by the maxima of the remainder, which shows $a \rhd (r-p) = m$. The p-update together with $m \leq r$ implies $p \leq r$.

By definition of $\rhd$ we get $m = a \rhd m$ from (I1) and hence $x + y \leq m$ with $x \in r \cdot \langle a_h | y$ can never occur. This implies that all $b[x]$ with $x \in r \cdot \langle a_h | y$ are calculated before $b[y]$. Thus we have that $b[x] = b'[x]$ holds for the update of $b[\cdot]$ in line 6 and we infer for $b[y]$:

$$\forall y \in m: \;\; b[y] = \left(\bigotimes\nolimits_{x \in r \cdot \langle a_h | y} b'[x]\right) \;\&\; \mathsf{t}(y) \;. \tag{U}$$

By the definition of $\otimes$ (cf. Equation (5.1) on page 60) and & we obtain point-wise predicates for $b[y]$ and $\mathsf{s}_{b[y]}$. For all $y \in m$ and $u, v \in r$ we have:

$$\begin{aligned} u\, b[y]\, v \;\Leftrightarrow\; & \left(\forall x \in s : u\, (\mathsf{s}_{b'[x]} + b'[x])\, v\right) \;\wedge \\ & \left((\exists x \in s : u\, b'[x]\, v) \;\vee\; u\, \mathsf{t}(y)\, v \right) \;, \end{aligned} \tag{U1}$$

$$u\, \mathsf{s}_{b[y]}\, v \;\Leftrightarrow\; \left(\forall x \in s : u\, \mathsf{s}_{b'[x]}\, v\right) \;\wedge\; u\, \mathsf{s}_{\mathsf{t}(y)}\, v \;, \qquad \text{where} \quad s = r \cdot \langle a_h | y \;. \tag{U2}$$

For $y \in (p - m)$ we have $b[y] = b'[y]$ and hence (I)' $\Rightarrow$ (I). Thus we will restrict our attention to all $y \in m$. As (I1) is already clear we just have to check (I2) and (I3). Therefore we show that (I)' $\Rightarrow$ (I2) $\wedge$ (I3) holds by establishing the following point-wise predicates (J2) ($\Leftrightarrow$ (I2)) and (J3) ($\Leftrightarrow$ (I3)),

$$\forall u, v \in r,\; y \in p: \;\; u\, b[y]\, v \Leftrightarrow u\, (q \cdot a \cdot q)\, v \;\wedge\; u\, \mathsf{s}_{b[y]}\, v \Leftrightarrow u\, (q \cdot \mathsf{s}_a \cdot q)\, v \;\wedge \tag{J2}$$

$$u\, \mathsf{t}(q)\, v \Rightarrow u\, b[y]\, v \;\wedge\; u\, (\neg q \cdot \mathsf{s}_{\mathsf{t}(q)} \cdot \neg q)\, v \Rightarrow u\, \mathsf{s}_{b[y]}\, v \;. \tag{J3}$$

where $q = r \cdot \langle a + 1_A | y$ as defined in (I).

For $u, v \in r$ we distinguish seven different cases, visualized in Figure 5.7. Each $x_i, x_j, x_k \in r \cdot \langle a_h | y$ in the figure represents a whole class of nodes having the same (depicted) properties. These x are direct successors of y and they are the indices for the $\otimes$-composition in Equation (U). The values of $b[y]$ and $\mathsf{s}_{b[y]}$ are given by the updates (U1) and (U2).

1) $u, v \in \langle a | x_i \;\wedge\; \neg(u\, (a + a^{-1})\, v)$ with $x_i \in r \cdot \langle a | y$: This means both u and v are above x_i (which is above y) in the Hasse diagram and $\neg(u\, (q \cdot a \cdot q)\, v) \;\wedge\; \neg(u\, \mathsf{s}_{q \cdot a \cdot q}\, v)$ holds. We

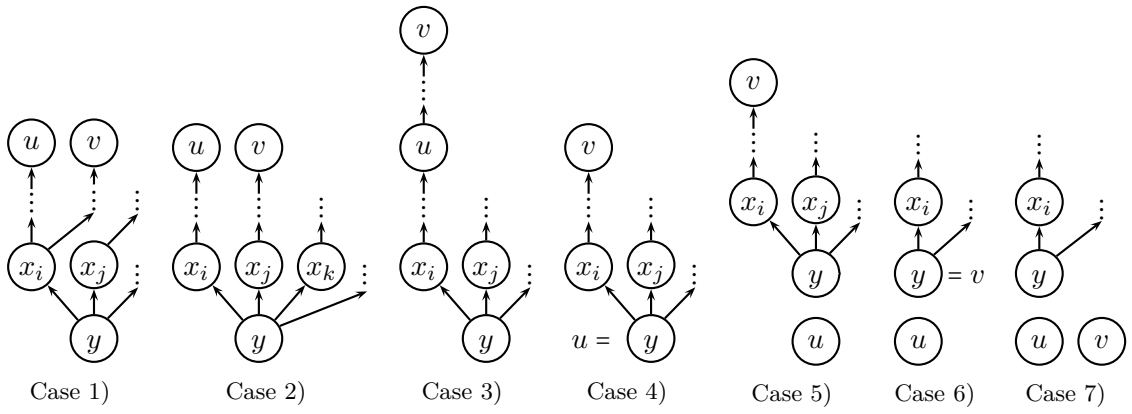


Figure 5.7: Partial Hasse diagrams for different positions of u, v with fixed y.

see that $u, v \in r \cdot \langle a + 1_A | x_i \leq q$ and by (I2)' we get that $\neg(u\, b'[x_i]\, v) \wedge \neg(u\, \mathsf{s}_{b'[x_i]}\, v)$, and hence (J2) follows.

As $u + v \leq q$, (J3) is clear, which also applies to the cases 2–4).

2) Otherwise, $u, v \in \langle a|y \;\wedge\; \neg(u\,(a + a^{-1})\, v)$: This means both u and v are above y in the Hasse diagram and $\neg(u\,(q \cdot a \cdot q)\, v) \wedge \neg(u\, \mathsf{s}_{q \cdot a \cdot q}\, v)$. By (I3)' we get that $\mathsf{t}(r \cdot \langle a + 1_A | u) \leq b'[x_i]$ and $\mathsf{t}(r \cdot \langle a + 1_A | v) \leq b'[x_j]$ holds. This yields (J2).

3) Otherwise, $u, v \in \langle a|y \;\wedge\; (u\,(a + a^{-1})\, v)$: W.l.o.g. we assume $(u\, a\, v)$. This means that v is above u and both are above y. Hence we have $(u\, b'[x_i]\, v)$ and $\neg(u\, \mathsf{s}_{b'[x_i]}\, v)$ from (I2)' and thus we get (J2).

4) Otherwise, $u, v \in q \;\wedge\; (u\,(a + a^{-1})\, v)$: W.l.o.g. we assume $u = y$ and thus $(u\, a\, v)$. With (I3)' we get $(u\, b'[x_i]\, v)$ and $\neg(u\, \mathsf{s}_{b'[x_i]}\, v)$. We conclude (J2).

5) Otherwise, $(u + v) \cdot \langle a|y \neq 0_A \;\wedge\; (u + v) \cdot \neg q \neq 0_A$. W.l.o.g. we assume $v \in \langle a|y$ and $u \notin \langle a|y$, hence $v \in q$ and $u \in \neg q$. We have (J2) analogous to case 4. As $u\, b[y]\, v$ is true and $u \notin \neg q$, (J3) also holds.

6) Otherwise, $(u + v) \cdot y \neq 0_A \;\wedge\; (u + v) \cdot \neg q \neq 0_A$. W.l.o.g. we assume $v = y$ and $u \notin q$. With (I2)' we get $u\, \mathsf{s}_{b'[x_i]}\, v$. With $u\, \mathsf{t}(y)\, v$ (because $y = v$) we retrieve $u\, b[y]\, v$. Thus we have (J2) and by $v \in q \wedge u \notin q$, (J3) also holds.

7) Otherwise, i.e. $u + v \leq \neg q$. Then $(u\, \mathsf{s}_{(q \cdot a \cdot q)}\, v)$ holds. Because of $u \neq y \neq v$ we have $(u\, \mathsf{s}_{\mathsf{t}(y)}\, v)$, hence we get $(u\, \mathsf{s}_{b[y]}\, v)$, thus (J2) and (J3).

Note that the validity of (I) in the cases 1–5) just follows from the $\otimes$-composition in (U). In the cases 6–7) the &-connected tuple preference ... & $\mathsf{t}(y)$ is also exploited.

When leaving the while loop we have $m = 0_A$ and hence $0_A = a \rhd (r - p) \;\wedge\; p \leq r$. By definition of $\rhd$ we get $r - p = 0_A \wedge p \leq r$ and thus $p = r$. Hence (I2) can be applied to all $b[x]$ for $x \in r$. Using the definition of $\otimes$ (similar to (U1), but without the & operator) in the assignment of b_{res} (line 11) we get that

$$\forall u, v \in r: \;\; u\, b_{\text{res}}\, v \;\Leftrightarrow\; u \left(\bigotimes\nolimits_{x \in r} b[x]\right) v \;\Leftrightarrow\; u\,(r \cdot a \cdot r)\, v$$

holds, which shows the claim. □

Remark 5.2.5. Note that the assignment $m \leftarrow a \rhd (r \cdot |a_h\rangle m)$ ensures that it is impossible to get $x + y \leq m \ \wedge \ (x\, a\, y)$ for any $x, y \in r$ in any run of the while loop. If we omit the maxima calculation in line 10, i.e. alter the assignment to $m = r \cdot |a_h\rangle m$, the algorithm is not correct anymore. In the proof of Theorem 5.2.4 the requirement that the $b[x]$ with $x \in r \cdot \langle a_h|y$ are calculated before $b[y]$ would by violated. As an example, consider the preference a in Figure 5.5. In the altered algorithm we get $m = x_3 + x_4$ in the second loop run, where $x_3 \in r \cdot \langle a_h|x_4$.

Within this section we now have proved all the results from Table 5.2, i.e., we classified the expressiveness of &, $\otimes$ and tuple/set preferences. The decomposition algorithms, implemented in rPref, are given in Appendix A.2. A script containing the implementation together with some more examples and a *preference decomposer* with a graphical user interface is available on the web [Roo15b].

5.3 Optimized Decomposition

The algorithms from the previous section generate much redundancy. For example, for the empty preference 0_A on a data set $r = x_1 + ... + x_n$ we get

$$\textsc{Decomp\_Pareto}(0_A, r) = \textsc{Decomp\_Tuple}(0_A, r) = \mathsf{t}(x_1) \otimes ... \otimes \mathsf{t}(x_n) \ .$$

Regarding the 0_A preference, all tuples are in just one layer, but the decompositions generate for every tuple in the data set r an own operand for a $\otimes$-composition. In general, the decomposition algorithms applied to layered preferences with many tuples per layer result in quite lengthy terms. We illustrate that in the following example.

Example 5.3.1. Let $b = \mathsf{t}(x_1 + x_2)$ on the data set $r = x_1 + ... + x_5$. First we obtain from the decomposition into a $\otimes$-composition of set preferences

$$\begin{aligned}\textsc{Decomp\_Pareto}(b, r) = \mathsf{t}(x_1) \otimes \mathsf{t}(x_2) \otimes \\ \mathsf{t}(x_1 + x_2 + x_3) \otimes \mathsf{t}(x_1 + x_2 + x_4) \otimes \mathsf{t}(x_1 + x_2 + x_5) \ .\end{aligned}$$

Next, we apply the decomposition into $\{\&, \otimes\}$ and tuple preferences as given in Algorithm 5.1. We get

$$\begin{aligned}\textsc{Decomp\_Tuple}(b, r) = &(\ (\mathsf{t}(x_1) \otimes \mathsf{t}(x_2)) \,\&\, \mathsf{t}(x_3) \ ) \otimes \\ &(\ (\mathsf{t}(x_1) \otimes \mathsf{t}(x_2)) \,\&\, \mathsf{t}(x_4) \ ) \otimes \\ &(\ (\mathsf{t}(x_1) \otimes \mathsf{t}(x_2)) \,\&\, \mathsf{t}(x_5) \ ) \ .\end{aligned}$$

Both terms are much longer than $\mathsf{t}(x_1 + x_2)$. When considering the Hasse diagram of b, given in Figure 5.8, we see that x_1 is equivalent to x_2 in the sense that their sets of a-predecessors and a-successors are identical. Analogously x_3, x_4 and x_5 have the same successors, namely x_1 and x_2.

Our idea is to identify all nodes which are equivalent in this sense and then to apply our decomposition algorithms to these simpler graphs. We exemplify this idea in Figure 5.8.

In the remainder of this section we formalize this idea.

5.3.1 Elimination of Equivalent Nodes

In the following we give a formal definition of the elimination of equivalent nodes. The induced preference on the quotient set is called the *minimized preference*.

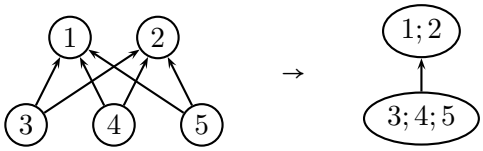


Figure 5.8: Preference b from Example 5.3.1 (left) and its simplified graph (right). A circled $i_1;...;i_k$ is short for the equivalence class $[\![x_{i_1}]\!] = ... = [\![x_{i_k}]\!]$.

Definition 5.3.2 (minimized preference)**.** Let a be a preference and r a data set. We define an equivalence relation $\sim_{a,r}$ where all tuples having the same predecessors and successors are in one equivalence class. Formally we define for all $u, v \in r$:

$$u \sim_{a,r} v \quad \Leftrightarrow_{df} \quad r \cdot |a\rangle u = r \cdot |a\rangle v \quad \wedge \quad r \cdot \langle a| u = r \cdot \langle a| v \ .$$

Let $r_{\min} =_{df} r/\sim_{a,r}$ be the quotient set of $\sim_{a,r}$. For $[\![u]\!], [\![v]\!] \in r_{\min}$ we define the minimized preference $a_{\min}$ on the equivalence classes by

$$[\![u]\!] \, a_{\min} \, [\![v]\!] \quad \Leftrightarrow_{df} \quad u \leq |a\rangle v \ ,$$

where tuples $[\![u]\!] \in r_{\min}$ on the quotient set contain in generally sets $u \leq r$ as representatives in the original set r.

Further we define $\mathsf{s}_{a_{\min}} = \sim_{a,r}$ as the SV relation of the minimized preference.

All other relational operations are lifted to the set of equivalence classes in the canonical way, especially we define $0_{\min} =_{df} 0_A/\sim_{a,r}$ and $1_{\min} =_{df} 1_A/\sim_{a,r}$. Relational composition with some other relation b on the original data set r (and not the quotient set) is canonically defined for $[\![u]\!] \in r_{\min}, v \in r$ by

$$[\![u]\!] \, (a_{\min} \cdot b) \, v \quad \Leftrightarrow_{df} \quad \exists w \in 1_A : [\![u]\!] \, a_{\min} \, [\![w]\!] \ \wedge \ w \, b \, v \ ,$$

and symmetrically we define $b \cdot a_{\min}$. This means that w and its associated equivalence class $[\![w]\!]$ establishes the connection between common relations and relations on equivalence classes w.r.t. $\sim_{a,r}$.

The set preference having an equivalence class as its argument, formally $\mathsf{t}([\![x]\!])$, is canonically defined by $[\![u]\!] \, \mathsf{t}([\![x]\!]) \, [\![v]\!] \Leftrightarrow_{df} u \leq |\mathsf{t}(x)\rangle v$ for all $[\![u]\!], [\![v]\!] \in r_{\min}$. □

This construction is very similar to constructing minimal automata by identifying equivalent states, cf. [Eil74].

The definition of $\sim_{a,r}$ is directly connected to the compatibility conditions for SV relations in Definition 3.2.1. This implies that $\sim_{a,r}$ is the maximal SV relation for a.

Lemma 5.3.3. *Definition 5.3.2 is well-formed, i.e., $a_{\min}$ is indeed a preference.*

Proof. First we show that for tuples $x, y \in [\![u]\!]$ with $u \in r_{\min}$, i.e., tuples in the same equivalence class, $\neg(x \, (a + a^{-1}) \, y)$ holds. This means that there are no better-than-relations within an equivalence class. We show this by contradiction. Assume that $(x \, (a + a^{-1}) \, y)$ holds. W.l.o.g. we presume $(x \, a \, y)$. This implies $x \in |a\rangle y$ by definition and $y \notin |a\rangle y$ by irreflexivity of a. This is a contradiction to $x, y \in [\![u]\!]$ by the definition of $[\![u]\!]$. With this, the definition of $a_{\min}$ and the property that a is a preference, we can simply verify that $a_{\min}$ is also a preference. By definition it is clear that $a_{\min} \cdot \sim_{a,r} = \sim_{a,r} \cdot a_{\min} = a_{\min}$ holds, hence $\sim_{a,r}$ is a valid SV relation for $a_{\min}$. □

5.3.2 Minimized Decomposition

We apply the elimination of equivalent nodes to the decomposition algorithms from Definition 5.2.1 and Algorithm 5.1.

Definition 5.3.4 (minimized decomposition)**.** Let a be a preference and $r \leq 1_A$ a finite data set. We define a $\otimes$-composition of set preferences where each set is upward closed w.r.t. $a_{\min} + 1_{\min}$, using the definitions for $r_{\min}$, $a_{\min}$ and $1_{\min}$ from Definition 5.3.2, by

$$\textsc{Dec\_Min}_1(a,r) =_{df} \bigotimes_{[\![x]\!] \in r_{\min}} \mathsf{t}(r \cdot \langle a_{\min} + 1_{\min} | [\![x]\!]) \ .$$

We define $\textsc{Dec\_Min}_2(a,r)$ as given in Algorithm 5.2.

Note that, even $\textsc{Dec\_Min}_2(a,r)$ is very similar to $\textsc{Decomp\_Tuple}(a,r)$ from Algorithm 5.1, it is not a decomposition into tuple preferences. It decomposes preferences into set preferences, which are connected by the $\{\&, \otimes\}$ operators. There each set corresponds to the equivalence class of a node in the preference graph, i.e., a tuple of the data set, or, in the case of minimized preferences, a subset of the data set. In contrast to this, $\textsc{Dec\_Min}_1(a,r)$ is a $\otimes$-composition of set preferences, where each set corresponds to more than one equivalence class in general.

The indices of the array $b[\cdot]$ in Algorithm 5.2 are the equivalence classes $r_{\min}$. The values of $b[\cdot]$ are preferences. The assignment $b[m] \leftarrow c$ for a non-empty set $m \subseteq r_{\min}$ is used as a shorthand notation for simultaneous assignments $b[[\![x]\!]] \leftarrow c$ for all $[\![x]\!] \in m$. We also assume that in all assignments the neutrality of $0_{\min}$ is used, implying that $b[[\![x]\!]] \leftarrow 0_{\min} \star c$ is executed as $b[[\![x]\!]] \leftarrow c$ for $\star \in \{\&, \otimes\}$.

Algorithm 5.2 Optimized preference decomposition into set preferences and $\{\&, \otimes\}$ using the definitions for $r_{\min}$ and $a_{\min}$ from Definition 5.3.2

Input: Preference to decompose a, data set r
Output: r-equivalent decomposition b_{res}

function $\textsc{Dec\_Min}_2(a,r)$
  $a_h \leftarrow r_{\min} \cdot (a_{\min} \sqcap (a_{\min})^2) \cdot r_{\min}$ // Hasse diagram of $a_{\min}$ on $r_{\min}$
  $b[r_{\min}] \leftarrow 0_{\min}$ // initialization of array b of preferences
  $m \leftarrow (a_{\min} \rhd r_{\min})$ // start traversing with equivalence classes of maxima
  while $m \neq 0_{\min}$ **do**
    for all $[\![y]\!] \in m$ **do** // pref. for y, collect and $\otimes$-compose successors,
      $b[[\![y]\!]] \leftarrow \big(\otimes_{[\![x]\!] \in \langle a_h | [\![y]\!]} b[[\![x]\!]]\big) \,\&\, \mathsf{t}([\![y]\!])$ // and add pref. on $[\![y]\!]$
    end for
    $b[\langle a_h | m] \leftarrow 0_{\min}$ // delete preferences of m-successors
    $m \leftarrow (a_{\min} \rhd |a_h\rangle m))$ // find a-maximal predecessors of m
  end while
  $b_{\text{res}} \leftarrow \otimes_{[\![x]\!] \in r_{\min}} b[[\![x]\!]]$; **return** b_{res} // $\otimes$-compose final preference
end function

Subsequently, we will show the correctness of these decomposition algorithms. Both are canonical transformations of the algorithms from Section 5.2, where the domain changes from r to the quotient set $r/{\sim_{a,r}}$. We will use the correctness proofs from that section in the following arguments.

Theorem 5.3.5. *Let a be a preference and r a data set. Then both* $\textsc{Dec\_Min}_1(a,r)$ *and* $\textsc{Dec\_Min}_2(a,r)$ *are r-equivalent to a.*

Proof. In Lemma 5.3.3 we have shown that $a_{\min}$ is indeed a preference. Hence the correctness of the decomposition algorithms, shown in Theorem 5.2.1 and Theorem 5.2.4, implies that $a_{\min}$ is $r_{\min}$-equivalent to $\text{DEC\_MIN}_1(a,r)$ and $\text{DEC\_MIN}_2(a,r)$. Additionally this implies that $\text{DEC\_MIN}_1(a,r)$ and $\text{DEC\_MIN}_2(a,r)$ are also well-defined preferences on $r_{\min}$. Immediately by Definition 5.3.2 we get that

$$r \cdot b \cdot r = r \cdot b_{\min} \cdot r \quad \text{for} \quad b \in \{a, \text{DEC\_MIN}_1(a,r), \text{DEC\_MIN}_2(a,r)\}$$

holds, i.e., the minimized preference is r-equivalent to non-minimized one. Finally we calculate, using the $r_{\min}$-equivalence of a and $\text{DEC\_MIN}_i(a,r)$,

$$\begin{aligned} r \cdot a \cdot r &= r \cdot a_{\min} \cdot r = r \cdot r_{\min} \cdot a_{\min} \cdot r_{\min} \cdot r \\ &= r \cdot r_{\min} \cdot \text{DEC\_MIN}_i(a,r) \cdot r_{\min} \cdot r = r \cdot \text{DEC\_MIN}_i(a,r) \cdot r \ , \end{aligned}$$

for $i \in \{1,2\}$. For the composition $r_{\min} \cdot r$ we apply Definition 5.3.2. The above calculation shows the claim. □

As a first example of these algorithms we apply the optimized decompositions to the simple layered preference from Example 5.3.1.

Example 5.3.6. Let $b = \mathsf{t}(x_1 + x_2)$ on $r = x_1 + \ldots + x_5$, as illustrated in Figure 5.8. For the optimized decompositions we get, where $\overset{r}{=}$ denotes r-equivalence:

$$\begin{aligned} \text{DEC\_MIN}_1(b,r) &= \mathsf{t}([\![x_1]\!]) \otimes \mathsf{t}([\![x_1]\!] + [\![x_3]\!]) \overset{r}{=} \mathsf{t}(x_1 + x_2) \otimes \mathsf{t}(r) \ , \\ \text{DEC\_MIN}_2(b,r) &= \mathsf{t}([\![x_1]\!]) \,\&\, \mathsf{t}([\![x_3]\!]) \overset{r}{=} \mathsf{t}(x_1 + x_2) \,\&\, \mathsf{t}(x_3 + x_4 + x_5) \ . \end{aligned}$$

The terms of $\text{DEC\_MIN}_i(b,r)$ for $i \in \{1,2\}$ are much shorter than the results of the decomposition methods $\text{DECOMP\_PARETO}(b,r)$ and $\text{DECOMP\_TUPLE}(b,r)$, shown in Example 5.3.1.

The optimized decomposition algorithms, implemented in rPref, are given in Appendix A.3.

In the following section we will apply this optimization to preferences on the power set of a domain. Such preferences typically result in large Hasse diagrams, where the optimized decomposition leads to much shorter preference terms.

To this end, we will motivate and introduce preferences on power sets in the following.

5.4 The Power Construction

In this section we consider the lifting of a preference to the power set of the domain. We study several variants of power constructions for strict orders from the literature. Then we apply the decomposition algorithms to them. Especially we consider how the optimized decomposition algorithms from the previous section work on the lifted preferences.

5.4.1 Motivation

Regarding the application of power set preferences, we are interested in constructing better-than-relations between sets starting from a given preference which is defined on the tuples that are contained in these sets. For example, assume that a user wants to rent a car and has the choice between two car rentals offering different choices of cars. The car fleets of these rental agencies are depicted in Figure 5.9. The user prefers powerful cars with low fuel consumption, hence the cars with id's 6, 8 and 7 are optimal for her. But none of the fleets contains all those three optimal choices. Assume further that both

rental agencies do not accept reservations for an individual car; they just guarantee that one of the cars of their fleet is available. The question arises, which car rental agency is superior for the user? Obviously none of these fleets is strictly better; some arrows point from fleet A to fleet B, some in the converse direction.

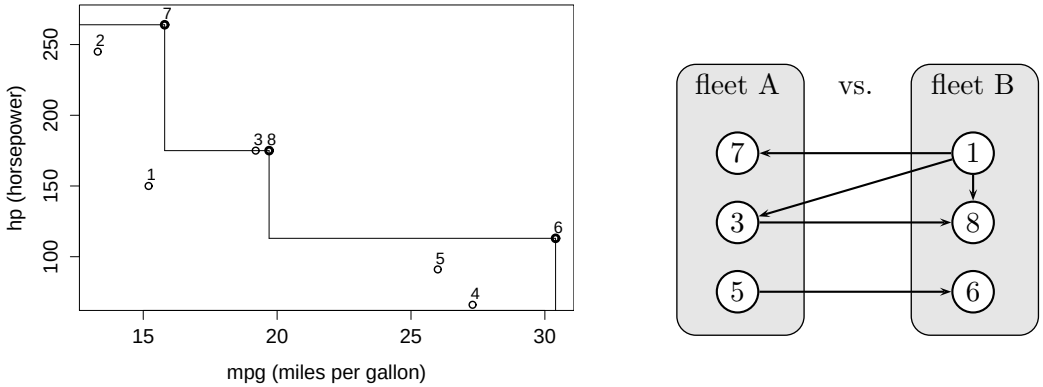


Figure 5.9: Left: Pareto optima of cars with low fuel consumption (i.e., high *(mpg)* value) and high horsepower *(hp)*. Right: Comparison of two car fleets with cars from the left diagram. Arrows point from worse to better objects in the sense of the Pareto optima.

To study this in detail, we have to consider preferences on sets, i.e., the extension of a preference to the power set of the domain. In the terminology of [BR01] this is called a *power construction*. Corresponding to the definitions in [BR01, Win85] there are three different possibilities of power constructions for strict orders, which have been used there in contexts other than preferences. In [BR01] different variants of programming semantics are discussed, where power constructions play an important role throughout. In [Win85], the focus lies on modelling non-deterministic computations. One of the power constructions for strict orders has already been used in the context of database preferences in [WBKH04].

We compare the unoptimized and the optimized versions of our algorithms on power set preferences. It turns out that this optimization leads to shorter preference terms especially on power set preferences. The implementation of the optimized algorithms with a graphical user interface and the comparative study at the end of this section is available on the web [Roo15b].

5.4.2 Definition and Properties of Power Set Preferences

First, we define the power construction for preferences on a given data set. According to [BR01, Win85] there are three natural extensions of a strict order to the power set of the domain which we will recapitulate in the following definition. We will assume a finite data set in the following, as we need finiteness to show that the power construction yields a preference.

Definition 5.4.1 (power set preference)**.** Let a be a preference on a finite data set r. We introduce preferences π_i^a for $i \in \{0,1,2\}$ on the power set $\mathcal{P}(r) = \{p \mid p \leq r\}$ by defining for all $u, v \in \mathcal{P}(r)$:

$$\begin{aligned} u\,\pi_0^a\,v \quad &\Leftrightarrow_{df} \quad v \neq 0_A \;\wedge\; \forall y \in v : \exists x \in u : x\,a\,y\ , \\ u\,\pi_1^a\,v \quad &\Leftrightarrow_{df} \quad u \neq 0_A \;\wedge\; \forall x \in u : \exists y \in v : x\,a\,y\ , \\ u\,\pi_2^a\,v \quad &\Leftrightarrow_{df} \quad u\,(\pi_0^a \sqcap \pi_1^a)\,v\ . \end{aligned}$$

Intuitively π_0 means that a set v is better than a non-empty set u, formally $(u\,\pi_0^a\,v)$, if every tuple in v dominates some tuple in u. For $(u\,\pi_1^a\,v)$ we require that every tuple in v is dominated by some tuple in u. Finally for $(u\,\pi_2^a\,v)$ both of these conditions have to be fulfilled, formally resulting in the intersection of π_0^a and π_1^a. In Figure 5.10 we show (partial) graphs of all power set preferences π_i^a $(i \in \{0,1,2\})$ for the "N-shaped" preference a in Figure 5.10(1). The power set preference π_1^a has already been used in the context of database preferences in [WBKH04], Definition 3.1.

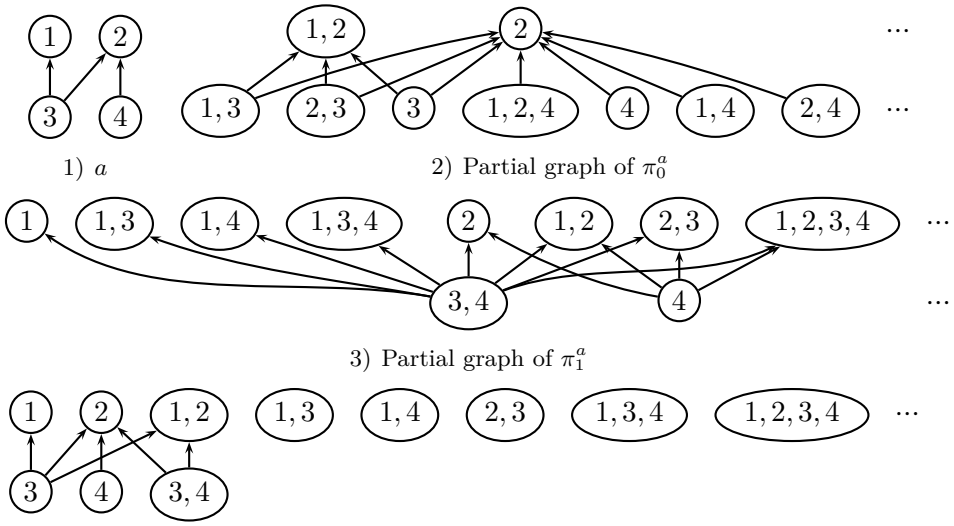


Figure 5.10: Preference a and a partial diagram of its induced power set preferences π_j^a for $j \in \{0,1,2\}$. A circled i is short for x_i and a circled $i_1,...,i_k$ short for $x_{i_1} + ... + x_{i_k}$.

Note that, without the second conjunct excluding empty sets in π_1^a and π_2^a, Definition 5.4.1 would lead to a relation not being irreflexive (and hence not a preference). Formally, let $u\,\pi_3^a\,v \Leftrightarrow \forall y \in v : \exists x \in v : x\,a\,y$, i.e., the definition of π_0^a without a special handling of empty sets. For this we get $(0_A\,\pi_3^a\,0_A)$, contradicting irreflexivity.

However, in the following we will exclude the empty set from the considered domain as it is not interesting for our application. For a data set r we define

$$\widehat{r} =_{df} \mathcal{P}(r)\backslash\{0_A\} = \{p \mid p \leq r \wedge p \neq 0_A\}\ ,$$

which will be the domain of power set preferences throughout the remainder of this section.

Subsequently, we introduce a quantifier-free representation of the predicates $(u\,\pi_i^a\,v)$, following the idea of [BR01], Theorem 2.31.

Corollary 5.4.2. *Let a be a preference on a data set r. For all $u,v \in \widehat{r}$ we have*

$$\begin{aligned} u\,\pi_0^a\,v \quad &\Leftrightarrow_{df} \quad v \leq \langle a|u\ , \\ u\,\pi_1^a\,v \quad &\Leftrightarrow_{df} \quad u \leq |a\rangle v\ . \end{aligned}$$

Proof. Immediately from the definition of $\widehat{r}$, $|a\rangle(\cdot)$ and $\langle a|(\cdot)$. □

Lemma 5.4.3. *Definition 5.4.1 is well-formed, i.e., π_i^a are indeed preferences.*

Proof. We will show this just for π_1^a. For π_0^a analogous arguments hold and for $\pi_2^a = \pi_0^a \sqcap \pi_1^a$ we exploit that preferences are preserved under intersection. We have to show that π_1^a is

irreflexive and transitive. For $u, v, w \in \widehat{r}$ we have, using Corollary 5.4.2, the isotony of the diamond and the transitivity of a,

$$u\,\pi_1^a\,v \;\wedge\; v\,\pi_1^a\,w \quad\Rightarrow\quad u \leq |a\rangle(|a\rangle w) \leq |a^2\rangle w \leq |a\rangle w \quad\Rightarrow\quad u\,\pi_1^a\,w\ ,$$

showing the transitivity of π_1^a. Next, we show irreflexivity by contradiction, i.e., we assume $(u\,\pi_1^a\,u)$ for $u \in \widehat{r}$. We calculate:

$$\begin{array}{ll} & u\,\pi_1^a\,u \\ \Leftrightarrow & \quad \{\![\text{ Corollary 5.4.2 }]\!\} \\ & u \leq |a\rangle u \\ \Leftrightarrow & \quad \{\![\text{ shunting } (p \cdot q \leq s \Leftrightarrow p \leq \neg q + s \text{ with } p = u, q = \neg|a\rangle u, s = 0_A) \ ]\!\} \\ & u - |a\rangle u \leq 0_A \\ \Leftrightarrow & \quad \{\![\text{ definition of } \rhd \ ]\!\} \\ & a \rhd u = 0_A\ . \end{array}$$

As $u \in \widehat{r}$ we have $u \neq 0_A$, i.e., u is a non-empty set. By the assumption of a finite r in Definition 5.4.1, we also have that $u \leq r$ is finite. But an empty maxima set $a \rhd u$ for a preference (strict order) a and a non-empty finite set u is a contradiction. Hence we have shown the irreflexivity of π_1^a. □

Now we have that the preference property is preserved under extending a preference to its power set. But for a layered preference a the power set preference π_2^a is not a layered preference in general. For example, let $r = x_1 + x_2$ and $(x_1\,a\,x_2)$ the only better-than-relation in a. By the definition of π_2^a we have $(x_1\,\pi_2^a\,x_2)$. The set $p = x_1 + x_2$ is incomparable to both x_1 and x_2, i.e., formally $(x_1\,\overline{\pi_2^a}\,p)$ and $(p\,\overline{\pi_2^a}\,x_2)$. Im summary we have

$$x_1\,\overline{\pi_2^a}\,p \;\wedge\; p\,\overline{\pi_2^a}\,x_2 \;\Rightarrow\; x_1\,\left(\overline{\pi_2^a}\right)^2\,x_2 \qquad \text{but} \qquad x_1\,\pi_2^a\,x_2\ .$$

Hence the negative transitivity of π_2^a is violated and thus it is not a layered preference.

Note that on $\mathcal{P}(r)$ the empty set 0_A is incomparable to all sets in $\widehat{r}$ w.r.t. π_i^a for all $i \in \{0, 1, 2\}$. On $\widehat{r}$, the power set preferences π_0^a and π_1^a preserve layered preferences, as we show now.

Lemma 5.4.4. *Let a be a layered preference and r a data set. Then, the preferences $\widehat{r} \cdot \pi_0^a \cdot \widehat{r}$ and $\widehat{r} \cdot \pi_1^a \cdot \widehat{r}$ are also layered preferences.*

Proof. Let a be negatively transitive, i.e., $(\overline{a})^2 \leq \overline{a}$. We have to show that the preferences $\widehat{r} \cdot \pi_i^a \cdot \widehat{r}$ are also negatively transitive. We show this for $b = \widehat{r} \cdot \pi_1^a \cdot \widehat{r}$ and $u, v, w \in \widehat{r}$:

$$\begin{array}{ll} & u\,\overline{b}\,v \;\wedge\; v\,\overline{b}\,w \\ \Leftrightarrow & \quad \{\![\text{ definition of } \overline{b} \ ]\!\} \\ & \neg(u\,(\widehat{r} \cdot \pi_1^a \cdot \widehat{r})\,v) \;\wedge\; \neg(v\,(\widehat{r} \cdot \pi_1^a \cdot \widehat{r})\,w) \\ \Leftrightarrow & \quad \{\![\ u, v, w \in \widehat{r}, \text{ definition of } \pi_a^1 \text{ and moving negation inside }]\!\} \\ & u = 0_A \;\vee\; \exists x \in u : \forall y \in v : x\,\overline{a}\,y \;\wedge\; v = 0_A \;\vee\; \exists x' \in v : \forall y' \in w : x'\,\overline{a}\,y' \\ \Leftrightarrow & \quad \{\![\ u, v \in \widehat{r}, \text{ hence } u \neq 0_A \text{ and } v \neq 0_A \ ]\!\} \\ & \exists x \in u : \forall y \in v : x\,\overline{a}\,y \;\wedge\; \exists x' \in v : \forall y' \in w : x'\,\overline{a}\,y' \\ \Rightarrow & \quad \{\![\text{ specialization } y = x', \text{ reorganization of quantifiers }]\!\} \\ & \exists x \in u : \exists x' \in v : \forall y' \in w : x\,\overline{a}\,x' \;\wedge\; x'\,\overline{a}\,y' \\ \Leftrightarrow & \quad \{\![\text{ definition of } (\cdot)^2 \ ]\!\} \end{array}$$

$$
\begin{aligned}
& \exists x \in u : \forall y' \in w : x\,(\overline{a})^2\,y' \\
\Rightarrow \quad & \{\![\text{ logic and negative transitivity of } a \,]\!\} \\
& u = 0_A \ \vee \ \exists x \in u : \forall y' \in w : x\,\overline{a}\,y' \\
\Leftrightarrow \quad & \{\![\text{ moving negation outside, definition of } \pi_a^1 \text{ and } u, w \in \widehat{r} \,]\!\} \\
& \neg(u\,(\widehat{r} \cdot \pi_1^a \cdot \widehat{r})\,w) \\
\Leftrightarrow \quad & \{\![\text{ definition of } \overline{b} \,]\!\} \\
& u\,\overline{b}\,w \ .
\end{aligned}
$$

For $\widehat{r} \cdot \pi_0^a \cdot \widehat{r}$ an analogous argument holds. Thus for $i \in \{0,1\}$, the preference $\widehat{r} \cdot \pi_i^a \cdot \widehat{r}$ is negatively transitive and hence a layered preference. □

Note that the quantifier-free formulation of π_i^a in Corollary 5.4.2 just applies to the predicates $(u\,\pi_i^a\,v)$. A point-free definition of π_i^a (and thus potentially leading to a point-free proof of the above theorem) would require much more machinery as we would need the power construction of the composition $\cdot$, the power construction of the inverse image $|\cdot\rangle$, etc. This is beyond the scope of this section.

In the following we consider the decomposition of the minimized power set preferences $(\pi_i^a)_{\min}$ for a given preference a on the data set r. Throughout we will use $\widehat{r}$ as domain for the power set preferences and their Hasse diagrams.

5.4.3 Examples for the Minimized Decomposition on Power Sets

Example 5.4.5. Reconsider the N-shaped preference a from Figure 5.10(1). We apply the elimination of equivalent nodes from Definition 5.3.2 to the power set preferences π_i^a for $i \in \{0,1,2\}$.

Consider π_1^a and its minimized variant $(\pi_1^a)_{\min}$, illustrated in Figure 5.11. The diagram of $(\pi_1^a)_{\min}$ is isomorphic to the original preference a. For the equivalence classes of $\sim_{\pi_1^a, r}$ we get

$$
\begin{aligned}
[\![x_1]\!] &= \{x_1 + y \mid y \leq x_3 + x_4\} \ , \\
[\![x_2]\!] &= \{x_2 + y \mid y \leq x_1 + x_3 + x_4\} \ , \\
[\![x_3]\!] &= \{x_3\} \ , \\
[\![x_4]\!] &= \{x_3, x_3 + x_4\} \ .
\end{aligned}
$$

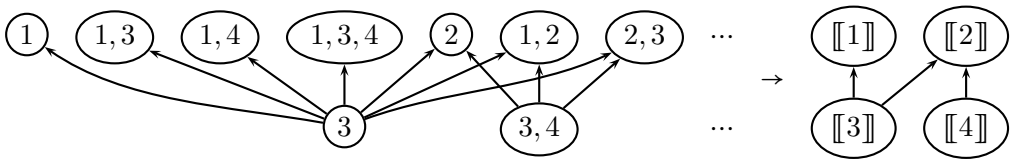


Figure 5.11: Partial diagram of preference π_1^a (left) and diagram of $(\pi_i^a)_{\min}$ for $i \in \{0,1\}$ (right), where a circled $[\![i]\!]$ is short for $[\![x_i]\!]$.

By definition of π_1^a, only the existence of dominating elements (here x_1 and x_2) in a set v is of interest to determine if $(u\,\pi_1^a\,v)$ holds. For example, x_1 is dominating x_3 w.r.t. a. Hence all sets v with $x_1 \in v$ are better than x_3. Consequently, sets like x_1, $x_1 + x_4$ and $x_1 + x_3$ belong to the same equivalence class w.r.t. $\sim_{\pi_1^a, r}$.

According to decomposition methods from Definition 5.3.4 and Algorithm 5.2 we get for the decomposed preference terms

$$\text{DEC\_MIN}_1(\pi_1^a, r) = \mathsf{t}([\![x_1]\!]) \otimes \mathsf{t}([\![x_2]\!]) \otimes \mathsf{t}([\![x_1]\!] + [\![x_2]\!] + [\![x_3]\!]) \otimes \mathsf{t}([\![x_1]\!] + [\![x_4]\!]) \ ,$$
$$\text{DEC\_MIN}_2(\pi_1^a, r) = ((\mathsf{t}([\![x_1]\!]) \otimes \mathsf{t}([\![x_2]\!])) \,\&\, \mathsf{t}([\![x_3]\!])) \otimes (\mathsf{t}([\![x_2]\!]) \,\&\, \mathsf{t}([\![x_4]\!])) \ .$$

Next we consider the power set preference π_0^a. The Hasse diagram of $(\pi_0^a)_{\min}$ has the same structure as $(\pi_1^a)_{\min}$ but the evaluation of the equivalence classes w.r.t. $\sim_{\pi_0^a, r}$ yields a different result:

$$[\![x_1]\!] = \{x_1,\ x_1 + x_2\} \ ,$$
$$[\![x_2]\!] = \{x_2\} \ ,$$
$$[\![x_3]\!] = \{x_3 + y \mid y \leq x_1 + x_2 + x_3\} \ ,$$
$$[\![x_4]\!] = \{x_4 + y \mid y \leq x_1 + x_2\} \ .$$

The equivalence relation for π_0^a is isomorphic to that of π_1^a and the isomorphism is given by the mapping $\phi : r \to r$ with

$$\phi(x_1) = x_4,\ \phi(x_4) = x_1,\ \phi(x_2) = x_3,\ \phi(x_3) = x_2 \ .$$

Formally it holds that $\sim_{\pi_1^a, r} = \sim_{\pi_0^a, \phi(r)}$. But note that this isomorphism does not change the better-than-relations, the graph of π_0^a is still the same as in Figure 5.11 (right). Only the existence of dominated tuples in a set is relevant to determine sets which are better w.r.t. π_0^a. For example, we can add any tuple to x_3 and this set is still worse than x_2, formally $(u\, \pi_0^a\, x_2)$ holds if $x_3 \in u$.

Finally we consider $\pi_2^a = \pi_0^a \sqcap \pi_1^a$. Now for $(u\, \pi_2^a\, v)$ with $u, v \in \widehat{r}$ we require that all tuples in u are dominated by tuples in v, and simultaneously the tuples in v have to dominate those of u.

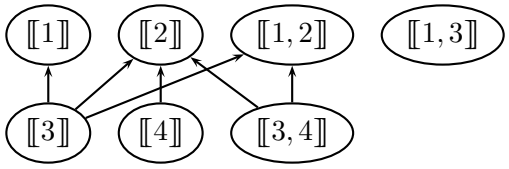


Figure 5.12: Diagram of preference $(\pi_2^a)_{\min}$. A circled $[\![i_1, ..., i_k]\!]$ is short for $[\![x_{i_1} + ... + x_{i_k}]\!]$.

This changes the Hasse diagram of π_2^a, depicted in Figure 5.12, compared to that of π_0^a and π_1^a (Figure 5.11). Each equivalence class in $r_{\min} - [\![x_1 + x_3]\!]$ forms an equivalence class of its own w.r.t. $\sim_{\pi_2^a, r}$. Their better-than-relations are identical to the non-minimized variant depicted in Figure 5.10(4). The class $[\![x_1 + x_3]\!]$, collecting the incomparable tuples w.r.t. π_2^a, can be formally characterized by

$$u \in [\![x_1 + x_3]\!] \quad \Leftrightarrow \quad u \sqcap (x_1 + x_2) \neq 0_A \ \wedge \ u \sqcap (x_3 + x_4) \neq 0_A \ .$$

This formalizes that all sets containing dominated nodes as well as dominating nodes are incomparable to all other sets w.r.t. π_2^a.

In the example above the Hasse diagrams of π_i^a for $i = 0$ and $i = 1$ are isomorphic to that of a. This is not always the case as we will show in the following example.

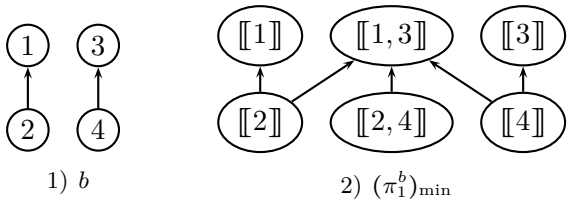


Figure 5.13: Preference b and its minimized induced power set preference $(\pi_1^b)_{\min}$.

Example 5.4.6. Consider the preference $b = (\mathsf{t}(x_1) \,\&\, \mathsf{t}(x_2)) \otimes (\mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4))$ on the data set $r = x_1 + \ldots + x_4$ and its minimized power set preference $(\pi_1^b)_{\min}$, depicted in Figure 5.13. For the preference π_1^b we get the decompositions

$$\begin{aligned}\textsc{Dec\_Min}_1(\pi_1^b, r) = {} & \mathsf{t}(\llbracket x_1 \rrbracket) \otimes \mathsf{t}(\llbracket x_1 + x_3 \rrbracket) \otimes \mathsf{t}(\llbracket x_3 \rrbracket) \otimes \mathsf{t}(\llbracket x_1 + x_3 \rrbracket + \llbracket x_2 + x_4 \rrbracket) \otimes \\ & \mathsf{t}(\llbracket x_1 \rrbracket + \llbracket x_1 + x_3 \rrbracket + \llbracket x_2 \rrbracket) \otimes \mathsf{t}(\llbracket x_3 \rrbracket + \llbracket x_1 + x_3 \rrbracket + \llbracket x_4 \rrbracket) \ ,\end{aligned}$$

$$\begin{aligned}\textsc{Dec\_Min}_2(\pi_1^b, r) = {} & (\ (\mathsf{t}(\llbracket x_1 \rrbracket) \otimes \mathsf{t}(\llbracket x_1 + x_3 \rrbracket)) \,\&\, \mathsf{t}(\llbracket x_2 \rrbracket)\ ) \otimes \\ & (\ (\mathsf{t}(\llbracket x_3 \rrbracket) \otimes \mathsf{t}(\llbracket x_1 + x_3 \rrbracket)) \,\&\, \mathsf{t}(\llbracket x_4 \rrbracket)\ ) \otimes \\ & (\ \mathsf{t}(\llbracket x_1 + x_3 \rrbracket) \,\&\, \mathsf{t}(\llbracket x_2 + x_4 \rrbracket)\ ) \ .\end{aligned}$$

The resulting terms in this example may still look quite complex. Still, the minimization is an advantage in many cases as we will underline in the quantitative comparison in the following.

5.4.4 Quantitative Comparison of the Decomposition Approaches

Now we compare the complexity of the generated terms by counting operands and operators. We define two measures for the complexity of a preference term:

1. The number of Boolean preferences in a complex preference term, where all base preferences are Boolean.

2. The number of $\otimes$-operators in a complex preference term.

The number of $\otimes$-operators is the main factor for the computational complexity of the preference evaluation, i.e., the determination of the maxima $a \rhd r$ for a given data set r. The costs to evaluate a $\otimes$-chain $a_1 \otimes \ldots \otimes a_n$ of layered preferences a_i quickly increases with the length n. The &-operator preserves layered preferences, cf. Corollary 5.1.4, whereas $\otimes$ does not. Hence the number of $\otimes$-operators is more important than the number of &-operators.

Subsequently, we formally define both of these complexity measures. There we assume a to be interpreted as a *preference term*, as it is represented in a preference implementation like rPref. As terms like $\mathsf{t}(x)$ and $\mathsf{t}(x) \otimes \mathsf{t}(x)$ are formally equivalent because of idempotency of $\otimes$, the subsequent definition would not be sound for abstract preference objects. The "=" operator is interpreted as *term equivalence* in the context of the following definition.

Definition 5.4.7 (preference term measures)**.** Let a be a preference term. We define $|a|_{\mathsf{t}}$ to be the number of Boolean preferences in a and $|a|_{\otimes}$ the number of $\otimes$ operators in a.

Formally we define them recursively by

$$|a|_{\mathtt{t}} =_{df} \begin{cases} |b|_{\mathtt{t}} + |c|_{\mathtt{t}} & \text{if } a = b \star c \text{ with } \star \in \{\otimes, \&\} \ , \\ 1 & \text{if } a = \mathtt{t}(\cdot) \ , \\ \text{undefined} & \text{otherwise} \ , \end{cases}$$

and

$$|a|_{\otimes} =_{df} \begin{cases} 1 + |b|_{\otimes} + |c|_{\otimes} & \text{if } a = b \otimes c \ , \\ |b|_{\otimes} + |c|_{\otimes} & \text{if } a = b \,\&\, c \ , \\ 0 & \text{if } a = \mathtt{t}(\cdot) \ , \\ \text{undefined} & \text{otherwise} \ . \end{cases}$$

For example, consider the preference $a = \mathtt{t}(x_1) \otimes (\mathtt{t}(x_2) \,\&\, \mathtt{t}(x_3)) \otimes \mathtt{t}(x_4)$. We get $|a|_{\mathtt{t}} = 4$ and $|a|_{\otimes} = 2$.

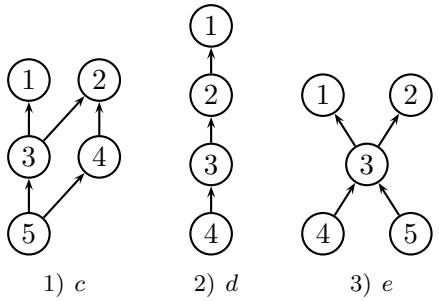


Figure 5.14: Hasse diagrams of preferences c, d, e used in the experiments.

Next to the preference a from Figure 5.10 and b from Figure 5.13 we consider c, d, e as depicted in Figure 5.14. The preference c is similar to the N-shaped preference a, but contains an additional tuple x_5 which is worse than all other tuples. Like a and b the preference c is also not a layered preference. The preferences d and e are layered preferences. We have chosen these examples, as they contain typical sub-graphs of larger preferences. They allow studying the principal effects, e.g., term complexity depending on the number of tuples, for some typical parallel and serial constructions of preferences.

In Table 5.3 we summarize the term complexity for each decomposition method, each preference ($a, ..., e$ from Figures 5.10, 5.13 and 5.14) and the power set preferences $\pi_i^{(\cdot)}$ for $i = 1$ and $i = 2$. We omitted $i = 0$ as this is very similar to $i = 1$ and hence the complexity is nearly the same. For the preferences a, b, d, e it is even exactly the same, because the Hasse diagrams of $\pi_0^{(\cdot)}$ and $\pi_1^{(\cdot)}$ are symmetric for each of these preferences, cf. the isomorphism shown in Example 5.4.5. All results have been retrieved with the help of an R script available online [Roo15b].

5.4.5 Discussion of the Results

For the non-minimized Pareto decompositions $(\cdot)_P$ in Table 5.3 we get in all cases $|(\cdot)_P|_{\mathtt{t}} = 2^i - 1$ and $|(\cdot)_P|_{\otimes} = 2^i - 2$ for the data set r_i, $i \in \{4, 5\}$ having the cardinality $|r_i| = i$. The data set $\widehat{r_i}$ has the cardinality $|\widehat{r_i}| = 2^i - 1$ for $i \in \{4, 5\}$. For every tuple from the data set one $\mathtt{t}(\cdot)$ operand is generated and then all operands are $\otimes$-composed. Hence, these values for $|\cdot|_{\mathtt{t}}$ and $|\cdot|_{\otimes}$ follow directly from the construction of the Pareto decomposition.

Analogously to these results, we get $|(\cdot)_{M1}|_{\mathtt{t}} = |(\widehat{r_i})_{\min}|$ and $|(\cdot)_{M2}|_{\otimes} = |(\widehat{r_i})_{\min}| - 1$ for the data set r_i and its minimization $(\widehat{r_i})_{\min}$ w.r.t. the given preference. We do not have a

Table 5.3: Quantitative comparison of different preference decompositions of power set preferences. We abbreviate $(\cdot)_P$ = DECOMP_PARETO$(\cdot, \widehat{r})$, $(\cdot)_T$ = DECOMP_TUPLE$(\cdot, \widehat{r})$, $(\cdot)_{M1}$ = DEC_MIN$_1(\cdot, \widehat{r})$, $(\cdot)_{M2}$ = DEC_MIN$_2(\cdot, \widehat{r})$ and $r_4 = x_1 + ... + x_4$, $r_5 = r_4 + x_5$.

Pref.	r	$\lvert(\cdot)_P\rvert_\mathsf{t}$	$\lvert(\cdot)_P\rvert_\otimes$	$\lvert(\cdot)_T\rvert_\mathsf{t}$	$\lvert(\cdot)_T\rvert_\otimes$	$\lvert(\cdot)_{M1}\rvert_\mathsf{t}$	$\lvert(\cdot)_{M1}\rvert_\otimes$	$\lvert(\cdot)_{M2}\rvert_\mathsf{t}$	$\lvert(\cdot)_{M2}\rvert_\otimes$
π_1^a	r_4	15	14	31	27	4	3	5	2
π_2^a	r_4	15	14	18	14	7	6	10	6
π_1^b	r_4	15	14	23	19	6	5	8	4
π_2^b	r_4	15	14	15	11	7	6	7	3
π_1^c	r_5	31	30	119	111	5	4	6	2
π_2^c	r_5	31	30	53	45	12	11	21	14
π_1^d	r_4	15	14	75	63	4	3	4	0
π_2^d	r_4	15	14	21	17	6	5	7	4
π_1^e	r_5	31	30	303	287	3	2	3	0
π_2^e	r_5	31	30	54	44	6	5	7	3

closed formula to get the number of equivalence classes, we just obtain the upper bound $|(\widehat{r_i})_{\min}| \leq |\widehat{r_i}| = 2^i - 1$.

For the preferences π_1^d and π_1^e we get $|(\cdot)_{M2}|_\otimes = 0$. This means that the minimized decomposition into $\{\&, \otimes\}$ and set preferences does not contain any $\otimes$-operator. Hence π_1^d and π_1^e are &-chains of set preferences. This is what we expect from Lemma 5.4.4 showing that the power construction preserves layered preferences. Those can be expressed by &-chains of set preferences, as shown in Lemma 5.1.9. Hence we do not need a $\otimes$-operator for the minimized decomposition DEC_MIN$_2$.

For all the other quantitative results we have no such "obvious" explanation. We consider this quantitative summary as an empirical result, giving some evidence that the minimization according to the $\sim_{(\cdot),r}$ equivalence relation is quite useful when decomposing power set preferences.

5.5 Complexity of the Decomposed Preference Terms

We have seen in the previous section that the resulting term length of the preference decomposition can increase quickly. We have introduced an optimized decomposition and evaluated it empirically for power set preferences.

Now we want to study the term complexity measures for the preference decomposition from a more theoretical perspective. This means we search for theoretical upper bounds and examples of concrete preferences, where these bounds are reached by order of magnitude.

As discussed in the previous section, for a data set r, we get for the term complexity measures in the case of the decomposition into $\otimes$-connected Boolean preferences,

$$|\text{DECOMP\_PARETO}(\cdot, r)|_\mathsf{t} = |r|, \quad |\text{DECOMP\_PARETO}(\cdot, r)|_\otimes = |r| - 1 \ .$$

Hence the complexity for this kind of decomposition just depends on the cardinality of the data set and not on the preference. From the empirical study in Table 5.3 we see that its optimized variant DEC_MIN$_1$ mostly obtains a bit smaller values for these measures. In any case, $|r|$ and $|r| - 1$ are upper bounds for $|\text{DEC\_MIN}_1(\cdot, r)|_\mathsf{t}$ and $|\text{DEC\_MIN}_1(\cdot, r)|_\otimes$, which is clear by construction of the equivalence classes within $r_{\min}$.

As the measures $|\cdot|_\mathsf{t}$ and $|\cdot|_\otimes$ are much higher for the results of DECOMP_TUPLE, and also much more shrinking for the optimized variant DEC_MIN$_2$, cf. Table 5.3, we will restrict

our attention to this kind of decomposition in the the following.

First, we will derive a theoretical upper bound of $|\text{DECOMP\_TUPLE}(\cdot, r)|_\mathsf{t}$, showing that terms grow at most factorially with the numbers of tuples in the data set. Next, we show an example of a preference, for which even the optimized variant of this decomposition, $\text{DEC\_MIN}_2(\cdot, r)$, reaches factorial term length.

5.5.1 An Upper Bound

Subsequently, we show that the term length $|\cdot|_\mathsf{t}$ for the decomposition into tuple preferences and $\{\&, \otimes\}$ grows not more than factorially.

Theorem 5.5.1. *Let a be a preference and $r_n = x_1 + ... + x_n$ a data set of cardinality n. For $c(n) =_{df} \text{DECOMP\_TUPLE}(a, r_n)$ we get for the number of $\mathsf{t}(\cdot)$ preferences in $c(n)$ that $|c(n)|_\mathsf{t} \leq (n+1)!$ holds.*

Proof. In the for-loop in line 6 of Algorithm 5.1 every tuple x is considered exactly once, cf. correctness proof in Theorem 5.2.4. Let $x^{(k)}$ be the k-th tuple being considered in line 6. Because a is a strict order and according to the m-updates, only the previously considered tuples $x^{(1)}, ..., x^{(k-1)}$ are potential successors of $x^{(k)}$. Hence there are at most $(k-1)$ direct successors of $x^{(k)}$, formally

$$|r \cdot \langle a_h | x^{(k)}| \leq k - 1 \ .$$

The preferences in $b[x^{(k)}]$ are constructed as a $\otimes$-composition of the $b[x]$ preferences with $x \in \langle a|x^{(k)}$, i.e., the successors of $x^{(k)}$. Finally a single tuple preference $\mathsf{t}(\cdot)$ is added after a &-operator. With $T(k)$ we denote the term length of the k-th constructed preference, formally,

$$T(k) =_{df} |b[x^{(k)}]|_\mathsf{t} \ .$$

According to the construction of the $b[x^{(k)}]$ as explained above, we get the following inequation as an upper estimate for $T(k)$.

$$T(k) \leq (k-1) \cdot T(k-1) + 1 \ .$$

Additionally we have $T(1) = 1$ as we see that $b[x^{(1)}] = \mathsf{t}(x^{(1)})$ holds for the first tuple, independently of the preference a. Then, by the upper estimate

$$T(k) = (k-1) \cdot T(k-1) + 1 \leq k \cdot T(k-1) \quad \text{for} \quad k \geq 2 \ ,$$

we get the recursive definition of the factorial $k!$, i.e., $T(k) \leq k!$.

Hence we have $|b[x^{(k)}]|_\mathsf{t} \leq k!$ which implies for $b_{\text{res}} = c(n)$, according to line 12 of the algorithm, that

$$|c(n)|_\mathsf{t} \leq n \cdot \sum_{i=1}^{n} |b[x^{(k)}]|_\mathsf{t} \leq n \cdot \sum_{i=1}^{n} k! \leq n \cdot n! \leq (n+1)!$$

holds. This shows the claim. □

Note that we used upper estimates of $|c(n)|_\mathsf{t}$ in several steps of the proof. In the next section we will see that a factorial growth of the term length w.r.t. the square root of the data set cardinality is indeed possible.

5.5.2 An Example for Factorial Growth

Now we show an example of a preference construction $a(k)$ on a data set $r(k)$, depending on an integer $k \in \{2, 3, ...\}$. We will show that the optimized decomposition into $\{\&, \otimes\}$ (i.e., DEC_MIN$_2$) of $a(k)$ leads to a preference term with a factorially growing term length w.r.t. k.

Subsequently, we formally define this preference, which we call *triangle preference* because of the shape of its Hasse diagram, visualized in Figure 5.15.

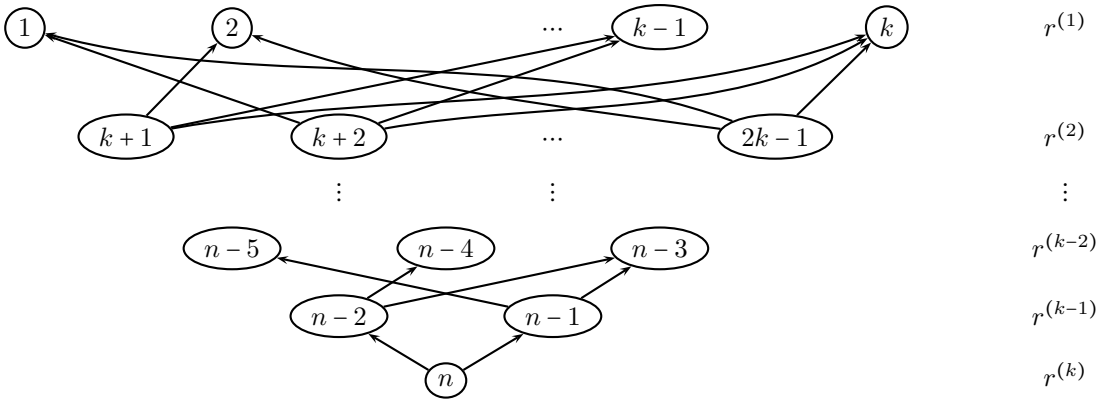


Figure 5.15: Hasse diagram of the triangle preference $a(k)$, as given in Definition 5.5.2, where a circled i is short for x_i. The sets $r^{(j)}$ contain all tuples x_i which are in the same row.

Definition 5.5.2 (triangle preference)**.** Let $k \geq 2$ be given. We specify the preference $a(k)$ in the following. First, we define

$$n(k) =_{df} k \cdot (k+1)/2 \ ,$$
$$r(k) =_{df} x_1 + ... + x_{n(k)} \ ,$$

where all x_i are distinct tuples. Next, we define the following sets for $l \in \{1, ..., k\}$:

$$r^{(l)} =_{df} \sum \left\{ x_m \;\middle|\; \sum_{j=0}^{l-1} (k-j) < m \leq \sum_{j=0}^{l} (k-j) \right\} \ .$$

The set $r^{(l)}$ corresponds to the l-th row in the diagram in Figure 5.5.2, counted from top to bottom. For example, we have $r^{(1)} = x_1 + ... + x_k$ and $r^{(k)} = x_{n(k)}$.

The better-than-relations of its Hasse diagram, formally $a_h(k) =_{df} a(k) \sqcap \overline{a(k)^2}$, are given by, for all $i, j \in \{1, ..., n(k)\}$,

$$x_i \, a_h(k) \, x_j \quad \Leftrightarrow_{df} \quad \exists l : x_i \in r^{(l)} \ \wedge \ x_j \in r^{(l+1)} \ \wedge \ (l = k-1 \ \vee \ j - i \neq k+1-l) \ . \qquad (5.3)$$

This means, that the rows $r^{(l)}$ and $r^{(l+1)}$ are fully connected except $(k-l)$ edges where $l \in \{1, ..., k-2\}$ denotes the considered row. Subsequently, we will explain which edges are missing.

Considering Figure 5.5.2, take a node y from the l-th row. The next bottom right node (e.g., x_{k+1} in the case of $y = x_1$) is not connected to y but to all other nodes from the row below y. Formally this is expressed by the condition $j - i \neq k + 1 - l$. The two bottom rows $r^{(k)} = x_{n(k)}$ and $r^{(k-1)} = x_{n(k-2)} + x_{n(k-1)}$ are fully connected to each other, which is formally expressed by $l = k - 1$. The actual preference $a(k)$ is given by the transitive hull of $a_h(k)$.

In Appendix A.4 we implemented the construction of the triangle preference based on rPref. Therefore we manually constructed the decomposition DECOMP_TUPLE$(a(k), r(k))$ as defined in Algorithm 5.1.

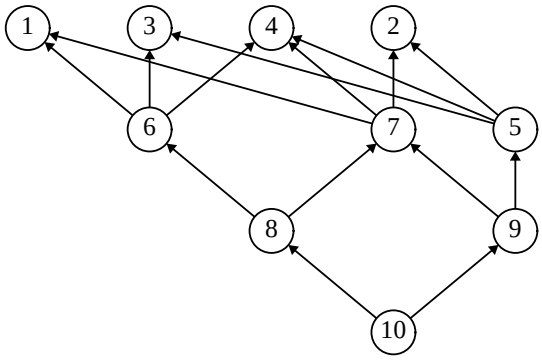


Figure 5.16: Hasse diagram of the triangle preference $a(k)$ with $k = 4$, plotted in rPref.

In Figure 5.16 we show the rPref output for the constructed preference $a(4)$. As $n(4) = 4 \cdot 5/2 = 10$, the data set is $r = x_1 + ... + x_{10}$. The appearance of this diagram slightly differs from the schema in Figure 5.15, as the layouting algorithm used in rPref arranged the two upper rows $r^{(1)}$ and $r^{(2)}$ not ascendingly.

In the following we state that the terms resulting from the optimized decomposition into $\{\&, \otimes\}$ grow factorially.

Theorem 5.5.3. *For the triangle preference $a(k)$, as given in Definition 5.5.2, we have for $k \geq 2$*

$$|\text{DEC\_MIN}_2(a(k), r(k))|_t \geq 2 \cdot (k-1)! \ .$$

Proof. Proof strategy: First we will show that $a(k)$ is constructed in such a way that the minimization does not bring any benefits, as any tuple resides in its own equivalence class. In the consequence we consider DECOMP_TUPLE instead of DEC_MIN$_2$. Then we will show the claim based on the construction of the decomposed preference.

According to the definition of $a(k)$ all tuples in the rows $r^{(1)}, ..., r^{(k-2)}$ have different sets of predecessors, cf. also Figure 5.15. For the first row $r^{(1)} = x_1 + .. + x_k$ we get

$$\begin{aligned} r(k) \cdot |a(k)\rangle x_1 &= r^{(2)} - x_{k+1} \ , \\ &\vdots \\ r(k) \cdot |a(k)\rangle x_{k-1} &= r^{(2)} - x_{2k-1} \ , \\ r(k) \cdot |a(k)\rangle x_k &= r^{(2)} \ . \end{aligned}$$

For the row $r^{(k-2)} = x_{n-5} + x_{n-4} + x_{n-3}$ we get

$$\begin{aligned} r(k) \cdot |a(k)\rangle x_{n-5} &= r^{(k-1)} - x_{n-2} = x_{n-1} \ , \\ r(k) \cdot |a(k)\rangle x_{n-4} &= r^{(k-1)} - x_{n-1} = x_{n-2} \ , \\ r(k) \cdot |a(k)\rangle x_{n-3} &= r^{(k-1)} = x_{n-1} + x_{n-2} \ , \end{aligned}$$

and for the rows between we get analogous results. The two nodes x_{n-2} and x_{n-1} of the row $r^{(k-1)}$ have the same predecessor (just x_n) but different successors:

$$\begin{aligned} r(k) \cdot \langle a(k)|x_{n-2} &= x_{n-4} + x_{n-3} \ , \\ r(k) \cdot \langle a(k)|x_{n-1} &= x_{n-5} + x_{n-3} \ . \end{aligned}$$

Finally the bottom row $r^{(k)}$ just contains x_n which is the only tuple in $r(k)$ having no predecessors.

According to the specification of the minimization, given in Definition 5.3.2, the equivalence classes are constructed as follows:

$$u \sim_{a,r} v \quad \Leftrightarrow_{df} \quad r \cdot |a\rangle u = r \cdot |a\rangle v \quad \wedge \quad r \cdot \langle a| u = r \cdot \langle a| v \ .$$

We have shown above that for $a = a(k)$ and $r = r(k)$ there do not exist two tuples $x, y \in r$ with $x \neq y$ whose predecessor/successor sets are identical. Hence for all equivalence classes $[\![x]\!]$ w.r.t. $\sim_{a,r}$ it holds that $[\![x]\!] = x$ for $x \in r(k)$, i.e., any tuple forms an equivalence class of its own.

It follows $a(k) = a(k)_{\min}$ and $r(k) = r(k)_{\min}$. In this case the algorithms DECOMP_TUPLE (Algorithm 5.1) and DEC_MIN$_2$ (Algorithm 5.2) are equivalent, what we see directly from the definitions of these algorithms.

Now we investigate DECOMP_TUPLE$(a(k), r(k))$. The algorithm annotates the tuples from top to bottom w.r.t. the diagram in Figure 5.15. More technically said, in the l-th run of the while loop we have $m = r^{(l)}$ and the $b[x]$ preferences for all $x \in r^{(l)}$ are calculated. Finally $b[x_n]$ is returned.

According to line 7 of the algorithm we calculate the $b[\cdot]$ values and their term lengths in the following. We get for the first row

$$\forall x \in r^{(1)} : b[x] = \mathsf{t}(x) \quad \Rightarrow \quad |b[x]|_{\mathsf{t}} = 1 \ .$$

For x_i, contained in of the rows $r^{(l)}$ with $l \in \{2, ..., k-1\}$ we get the following recursion formula, according to the algorithm and using Equation (5.3),

$$\forall i \in k+1, ..., n(k)-1 : \exists l : \ x_i \in r^{(l)} \ \wedge \\ b[x_i] = \Big(\bigotimes \{ \mathsf{t}(x_j) \mid x_j \in r^{(l-1)} \ \wedge \ j - i \neq k+1-l \} \Big) \,\&\, \mathsf{t}(x_i) \ .$$

Note that there always exists a unique l in the above formula, because $r^{(2)} + ... + r^{(k-1)}$ is a disjoint partition of $x_{k+1} + ... + x_{n(k)-1}$. There are $(k+2-l)$ tuples in the row $r^{(l-1)}$, whose $b[\cdot]$-values are all $\otimes$-operands in $b[x_i]$ despite the one element fulfilling $j - i = k+1-l$. Hence $b[x_i]$ with $x_i \in r^{(l)}$ is a $\otimes$-composition of $(k+1-l)$ operands, followed by $\& \, \mathsf{t}(x_i)$. For the term length the above formula implies

$$\forall i \in k+1, ..., n(k)-1 : \exists l : \ \ x_i \in r^{(l)} \ \wedge \ |b[x_i]|_{\mathsf{t}} = (k+1-l) \cdot |b[x_j]|_{\mathsf{t}} + 1 \ \text{ for } \ x_j \in r^{(l-1)} \ .$$

By omitting the summand 1 we get the lower estimate

$$|b[x_i]|_{\mathsf{t}} \geq (k+1-l) \cdot |b[x_j]|_{\mathsf{t}}$$

for the term length in row l, i.e., $x_i \in r^{(l)}$. For the final result $b[x_n]$ we obtain

$$b[x_n] = (b[x_{n-2}] \otimes b[x_{n-1}]) \,\&\, \mathsf{t}(x_n) \quad \Rightarrow \quad |b[x_n]|_{\mathsf{t}} = 2 \cdot |b[x_{n-1}]|_{\mathsf{t}} + 1 \ .$$

By induction over the rows $r^{(2)}, ..., r^{(k)}$ we get from the above equations that

$$\begin{aligned} |b[x_n]|_{\mathsf{t}} &\geq 2 \cdot (k+1-(k-1)) \cdot (k+1-(k-2)) \cdot ... \cdot (k+1-2) = \\ &= 2 \cdot 2 \cdot 3 \cdot ... \cdot (k-1) = \\ &= 2 \cdot (k-1)! \end{aligned}$$

holds. Hence we conclude that

$$|\text{DEC\_MIN}_2(a(k), r(k))|_{\mathsf{t}} = |\text{DECOMP\_TUPLE}(a(k), r(k))|_{\mathsf{t}} \geq 2 \cdot (k-1)! \ ,$$

which shows the claim. □

In the following corollary we show that the term length of the decomposition of $a(k)$ grows factorially w.r.t. the square root of the data set cardinality.

Corollary 5.5.4. *For the triangle preference $a(k)$ on a data set with cardinality $n(k)$, as given in Definition 5.5.2, we have for $k \geq 2$:*

$$|\text{DEC\_MIN}_2(a(k), x_1 + ... + x_{n(k)})|_t \geq 2 \cdot \left(\left\lfloor \sqrt{2 \cdot n(k)} \right\rfloor - 1\right)! \ .$$

Proof. We calculate:

$$\begin{array}{ll} & \mathsf{true} \\ \Rightarrow & \{\![\text{ definition of } n(k), \text{ arithmetic }]\!\} \\ & \sqrt{2 \cdot n(k)} = \sqrt{2 \cdot k \cdot (k+1)/2} = \sqrt{k^2 + k} \\ \Rightarrow & \{\![\text{ arithmetic }]\!\} \\ & k \leq \sqrt{2 \cdot n(k)} < k + 1 \\ \Leftrightarrow & \{\![\text{ definition of } \lfloor \cdot \rfloor \ ]\!\} \\ & \left\lfloor \sqrt{2 \cdot n(k)} \right\rfloor = k \end{array}$$

Substituting k by $\left\lfloor \sqrt{2 \cdot n(k)} \right\rfloor$ in Theorem 5.5.3 shows the claim. □

Table 5.4: Characteristics of the triangle preference, i.e., $|\text{DEC\_MIN}_2(a(k), r(k))|_t$ and lower bounds $2 \cdot (k-1)!$ for $k \in \{2, ..., 8\}$, retrieved with help of an R script.

k	$n(k)$	$2 \cdot (k-1)!$	$\lvert\text{DEC\_MIN}_2(a(k), r(k))\rvert_t$
2	3	2	3
3	6	4	7
4	10	12	19
5	15	48	67
6	21	240	307
7	28	1440	1747
8	36	10080	11827
9	45	80640	92467
10	55	725760	818227

In the proof of Theorem 5.5.3 we used lower estimates for the term length of $a(k)$ in several steps. In Table 5.4 we show the actual values for the term lengths of the triangle preference instances, which are slightly higher. These values were calculated with help of an R script available on the web [Roo15b].

5.5.3 Concluding Remarks

The preference decomposition into tuple preferences and $\{\&, \otimes\}$ is an important theoretical tool to characterize the expressiveness of preference query languages. But for practical applications the generated terms are in most cases much too complex. The optimization brings some benefits, e.g., for power set preferences, as shown in Section 5.4.4. But for our contrived example of the triangle preference, the decomposed preference remains unchanged under the minimization.

The simpler approach of the decomposition into $\otimes$-composed set preferences is a more reasonable approach for representing arbitrary preferences on small data sets. The length

of the constructed terms is restricted by the number of tuples in the data set. The operands are not simple tuple preferences but set preferences, containing potentially all tuples of the data set in their arguments.

A related problem is the search for a minimal decomposition in the following sense. For a given preference a and data set r we search for $b = b_1 \otimes ... \otimes b_n$ such that b is r-equivalent to a, all b_i are layered preferences and n is minimal. In [EP16] the authors present an algorithm which generates decompositions which are much shorter then our approach but not minimal. It is an open question if there exists a unique minimal decomposition and if there is an efficient algorithm, i.e., with polynomial run time, solving this problem.

CHAPTER 6

Implementations and Use Cases

Having introduced much theory in the previous parts of this thesis, we will dedicate this chapter to different tools and implementations related to database preferences. We will evaluate an automated theorem prover for showing a theorem from Chapter 3. We will show how the preference language and the preference evaluation algorithms can be implemented in R. We describe the approach for a rapid prototyping of preferences in R and the rPref package which became an official CRAN package. Regarding the preference decomposition from the previous chapter, this package served as an important toolbox for our research.

6.1 Using Prover9 for the Distributive Law

In this section we demonstrate how the proof of the distributive law for & and $\otimes$, given in Section 3.6, can be mechanised using Prover9 [McC05]. To this end we have to model the axioms of the join algebra, Definition 3.1.6, and the underlying abstract relation algebra. These axioms are the input for the automated theorem prover. Our goal is to prove that $a \,\&\, (b \star c) = (a \,\&\, b) \star (a \,\&\, c)$ holds for $\star \in \{\ltimes\otimes, \otimes\rtimes, \otimes\}$ and preferences a, b, c. In the following we will show this for the $\otimes\rtimes$ operator. The proof for $\ltimes\otimes$ is analogous to that. For $\otimes$ the claim follows directly from the two first ones, cf. the proof of Theorem 3.6.2. The following is an revised and extended version of [MR12a]. Compared to that report we could reduce the number of input files from four to two and we proved the theorem for general SV-semantics. The proof in [MR12a] only covers the special case $\mathsf{s}_x = 1_x$ for all preferences x.

6.1.1 Proof Strategy

The axiomatization in the prover input slightly differs from the definitions of the join algebra and the $\{\&, \otimes\}$ operators. An exact modelling of all axioms resulted in a problem of such complexity that the prover did not terminate within reasonable time. Therefore, we simplified the join algebra by removing the typing of 0, i.e., we assume $0_T = 0_{T'}$ for all types T, T' in the prover input. This is in accordance to the relational interpretation where the elements 0_T are represented by the empty set $\varnothing$, independent of the type T. This assumption simplifies modelling the 0-strictness of $\bowtie$. For $p :: T'$ we have formally, assuming a typed zero element, $0_T \bowtie p = 0_T \bowtie 0_{T'} = 0_{T \bowtie T'}$. In the relational setting, where

M is some set, this simplifies to $\varnothing \times M = \varnothing$. The same applies to our prover input.

As a small technical difference, we do not use explicit type assertions like $a :: T_a^2$ in our prover input, as the typename is not relevant. The only relevant point is, that every element has a type, besides the untyped 0 element. This can be realized by the axiom

$$\forall\, x:\ \ x \neq 0 \ \Rightarrow\ \exists\, T : x :: T\ . \tag{6.1}$$

corresponding to the first line of the prover input `x != 0 -> exists T (x typed T).`

Note that in Prover9 all variables beginning with the letter u, v, w, x, y, or z, are universally quantified. One can define one's own operators with a given binding strength. We use the following infix operators:

Prover-Input	mathematically
`a typed T`	$a :: T_a^{(2)}$
`a join b,`	$a \bowtie b$
`a prior b`	$a \,\&\, b$
`a rpar b`	$a \otimes\!\!> b$
`a + b`	$a + b$

There the binding strength is descending. In Prover9 this is stated as follows, where lower numbers indicate a tighter binding:

```
op(400, infix, "typed").
op(410, infix, "join").
op(500, infix, "prior").
op(510, infix, "rpar").
op(600, infix, "+").
```

We split the proof into the following two parts, introducing one intermediate step:

$$\begin{aligned}
& (a \,\&\, b) \otimes\!\!> (a \,\&\, c) \\
= \quad & \{\![\text{ input file 1 }]\!\} \\
& a \bowtie \top_b \bowtie a \bowtie \top_c \ + \ a \bowtie \top_b \bowtie \mathsf{s}_a \bowtie c \ + \\
& \mathsf{s}_a \bowtie (b + \mathsf{s}_b) \bowtie a \bowtie \top_c \ + \ \mathsf{s}_a \bowtie (b + \mathsf{s}_b) \bowtie \mathsf{s}_a \bowtie c \\
= \quad & \{\![\text{ input file 2 }]\!\} \\
& a \,\&\, (b \otimes\!\!> c)\ .
\end{aligned}$$

Proving the whole theorem in one step did not succeed within acceptable time. For both steps we just used proper subsets of the axioms of the join algebra in the input files.

6.1.2 Prover Input

Now we show the two input files for both steps of the proof. Both files are available on the web at [Roo15d]. Subsequently, we will explain the input files.

In the segment "abstract relation algebra" of the input file following, we introduce the type convention according to Equation (6.1) and the axioms of the typed abstract relation algebra from Sections 3.1.2 and 3.1.3. In "joins" we axiomatize the distributivity of $\bowtie$ over $+$, as required in Definition 3.1.6. The other axioms of the join algebra are not needed in this step. Adding all axioms would result in a too complex proving problem. In the segment "preferences" we axiomatize Prioritisation and Pareto with SV relations as given in Section 3.2. Finally, we denote the intermediate step. The goal is to show that the left hand side of the distributive law is equivalent to that.

Assumptions

```
% abstract relation algebra
% -------------------------

% all non-zero elements are typed
x != 0 -> exists T (x typed T).

% addition is associative, commutative and idempotent
(x + y) + z = x + (y + z).
x + y = y + x.
x + x = x.

% 0 is neutral element (0 is not typed!)
x + 0 = x.

% addition preserves type
x typed z & y typed z -> (x+y) typed z.

% subsumption order
x <= y <-> y = x + y.

% top is greatest element
x typed u -> x <= top(u).

% top distributes over join
top(x join y) = top(x) join top(y).

% top depends on the type
x typed u -> top(x) = top(u).

% joins
% -----

% distributivity of join over +
x join (y1 + y2) = x join y1 + x join y2.
(x1 + x2) join y = x1 join y + x2 join y.

% preferences
% -----------

% prioritisation
x prior y = x join top(y) + sv(x) join y.

% prioritisation is SV-preserving
sv(x prior y) = sv(x) join sv(y).

% right semi-Pareto
x rpar y = (x + sv(x)) join y.

% properties of SV relations
x typed u -> sv(x) typed u.

% intermediate step 1
% -------------------

step1 = ( (a     join top(b))      join (a     join top(c))   +
          (a     join top(b))      join (sv(a) join c)      ) +
        ( (sv(a) join (b + sv(b))) join (a     join top(c))   +
          (sv(a) join (b + sv(b))) join (sv(a) join c)      ).
```

Goals

```
(a prior b) rpar (a prior c) = step1.
```

Prover9 finds a proof in less than one second. For the second step we have the following input, where the segments "abstract relation algebra" and "preferences" are equivalent to the first input file. In the segment "joins" we denote some more axioms from the join algebra, Definition 3.1.6, as in the first step. To slightly simplify the input, we use axiom 3.1.6.7 of the join algebra in connection with Lemma 3.2.2, resulting in $\mathsf{s}_a \bowtie a = \mathsf{s}_a \sqcap a = 0_a$ for a preference a. Finally we formulate the goal, that the intermediate step is equivalent to the right hand side of the distributive law.

Assumptions

```
{  segments "abstract relation algebra" and "preferences" from step 1  }

% joins
% -----

% distributivity of join over +
x join (y1 + y2) = x join y1 + x join y2.
(x1 + x2) join y = x1 join y + x2 join y.

% join is associative, commutative and idempotent
(x join y) join z = x join (y join z).
x join y = y join x.
x join x = x.

% typing of join
x typed u & y typed v -> (x join y) typed (u join v).

% join on same type equals meet / compatibility of SV relation
x join sv(x) = 0.

% intermediate step 1
% -------------------

step1 = ( (a     join top(b))      join (a     join top(c))   +
          (a     join top(b))      join (sv(a) join c)      ) +
        ( (sv(a) join (b + sv(b))) join (a     join top(c))   +
          (sv(a) join (b + sv(b))) join (sv(a) join c)      ).
```

Goals

```
step1 = a prior (b rpar c).
```

Finding a proof for that input takes around five seconds. We also tried to prove the first step using the assumptions of the second step. This should be possible, as the input of the first step is a subset of that one of the second step. But we did not get a result within 60 seconds. Mainly the distributivity, associativity and commutativity of the $\bowtie$ operator generate a very large search tree, as there are many rewritings possible.

6.1.3 Evaluation

Compared to the previous version [MR12a], published together with [MRE12], we found a shorter way to prove the distributive law. We reduced the intermediate steps from three to one. But we are still far away from a fully automated proof, where we can thoughtless put all our axioms into the input. The assumption that 0 is untyped is a straightforward simplification, but is a step away from the type calculus. We tried to introduce a typed 0 for both steps, and for both modified input files Prover9 did not find a proof in reasonable time.

We had to carefully select the axioms for the input files to get results within acceptable time. Additionally, we needed to take care of the order of the axioms. For example, after

putting the block "intermediate step 1" before "joins" we did not get a result within 60 seconds. Before that reordering, this step took less than five seconds.

Instead of putting the law `x join sv(x) = 0` directly into the input file, we tried to derive it by the prover by stating requirement 3.1.6.7 and axiomatizing the $\sqcap$ operator. After this modification we did also not find a proof within 60 seconds.

The axiomatization of the join algebra offers a possibility to encode all the problems in this context in automated theorem provers. But for one-step proofs of non-trivial lemmas the structure is too complex.

6.2 Rapid Prototyping of Preferences and the rPref Package

In this section we describe the development of a rapid prototyping tool for preferences in the statistical computing language R [R C15]. The first results have been published in [RK13], describing the R-Pref script [Roo13]. In the following time, this idea lead to the rPref package [Roo15f]. This package was an important research tool for the preference decomposition problem, described in the previous chapter. The preference evaluation algorithms in rPref are implemented in C++ and thus quite fast. As the preference language in rPref is sticking close to the theory, we give some use cases, illustrating examples from the first chapters of this thesis.

6.2.1 Motivation

The preference constructs, as described in Section 2.2, have been evaluated within the research prototype *Preference SQL* [KEW11] in the beginning of our work. This system is a complex preference interpreter together with a query parser, an algebraic optimizer and many different algorithms. It operates on top of different database systems. Hence it is a prototyping approach which is conceptually quite near to a productive database system.

The problem of this approach is that new preference constructs cause changes in many parts of the system, at least in the parser and in the core system. Often some adaptations in the algorithms and the optimizer are additionally necessary to implement a new preference constructor. Because of this there was a growing demand for a small rapid prototyping system, which allows implementing new features without causing changes in many different parts of the code. Additionally, it should be possible to quickly evaluate and visualize the effects of new preference constructs. In the beginning of 2013 we started to develop such a prototyping system for preferences in R, called *R-Pref* [Roo13].

We decided to use the statistical programming language R because of several reasons. First, the concise and pragmatic way of coding in R is well-suited for rapid prototyping. In R, imperative and functional concepts work well together, and it has a dynamic type system. Next, preferences are often used as an alternative to statistical approaches in the application. For example, in the scope of recommender systems, preferences are used to explicitly model the knowledge of domain experts [REMK12]. In contrast, classical recommender systems are based on similarity, e.g., the user gets recommendations based on her previous choices. There classical statistical approaches are used, for which a language like R is predestined. Hence the use of R allows comparing preferences and classical approaches. Finally, the visualization functionality in R has supported us in many steps of our work. The visualization of preference graphs as well as Skyline diagrams makes it much easier to see the effect of preference selections in a concrete use case.

In the old R-Pref implementation, the maximum operator for preferences has been implemented directly in R. This has the advantage that code and formal definition are very

similar, cf. [RK13]. But it has the big disadvantage that the old R-Pref implementation is very slow. The preference selection, especially the top-k selection, could be used only for data sets with less than 1000 tuples to get a result in reasonable time. Thus we decided to implement all preference algorithms in C++ and connect them with R using the RCpp [Edd13] package. This allows a seamless integration of C++ and R, e.g., all data structures in R are wrapped into the corresponding classes of the STL (standard template library) in C++. Of source, the performance of this implementation cannot compete with the Exasol Exasolution database. But for "typical" R data sets with less then one million tuples, one mostly gets a result of a preference query in less than a few seconds.

The rPref package is part of the official R repository CRAN since 2014 and has been updated several times to integrate new constructs and algorithms. It offers both a rapid prototyping framework for preferences and efficient algorithms like the Scalagon algorithm [ERK15], cf. also Section 4.3. As another example, in [RWR+15] we showed how new preference constructs for preferences on hierarchies can be prototyped with rPref. In this context, prototyping means that the package source code remains untouched. We can formulate new constructs using the interface of the rPref package. Also the algorithms for the preference decomposition in Chapter 5 have been implemented on top of the rPref package. The R code of these algorithms is quite similar to the pseudo code in this thesis, cf. Appendices A.2 and A.3.

Subsequently, we will first give a more technical description of the package and the sketch the connection to the theory. After that we give some use cases, illustrating how rPref helped us to evaluate the theory as well as new preference concepts.

6.2.2 Description of the rPref Package

In the rPref package, preferences can be formulated using preference constructors very similar as in Section 2.2. Every constructor returns an R object, which has the type of a subclass of *preference*. The class hierarchy of all preference subclasses is given in Figure 6.1. For technical reasons, the actual class names of all these subclasses have the suffix "pref" (e.g., `emptypref`, `paretopref`, ...) which we omit here for sake of readability.

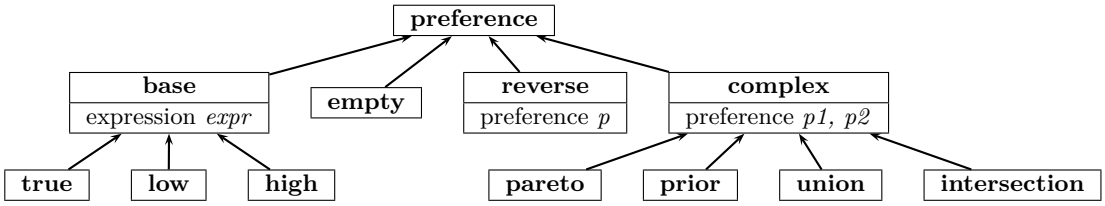


Figure 6.1: Class diagram of preferences in rPref.

The member variable *expr* for base preferences has the data type *expression* and represents an arbitrary R expression, modelling the ϕ-function from Definition 2.2.2 (numerical low and high preferences), or the ρ-predicate from Definition 2.2.4 (Boolean preferences), respectively.

In contrast to the theory there is a special class for the *empty* preference, representing the element 0_T for a given type T. Mathematically, a "constant" base preference like `low(0)` is identical to `empty()`. But we wanted to have an explicit neutral element for Pareto and Prioritisation, such that the preference interpreter of rPref can easily apply optimization rules like $a \otimes 0_a = a$.

Formally the reverse preference, prioritization and Pareto are all complex preferences. In rPref we distinguish between *reverse* as the only unary complex preferences and all

binary operators, which are subclasses of *complex*.

The R language allows overloading operators, hence we defined infix operators for the complex preferences very close to the formal notation. They are listed in Table 6.1, together with the corresponding query phrases in Preference SQL and Exasolution.

Table 6.1: Operators for complex preferences, where a, b are preferences.

rPref operator	formula	rPref class	Preference SQL	Exasolution Skyline
`-a`	a^{-1}	`reverse`	`a dual`	`reverse(a)`
`a * b`	$a \otimes b$	`pareto`	`a and b`	`a plus b`
`a & b`	$a \,\&\, b$	`prior`	`a prior to b`	`a prior to b`
`a + b`	$a + b$	`union`	`a disjoint union b`	*(not available)*
`a | b`	$a \sqcap b$	`intersect`	`a intersect with b`	*(not available)*

For the preference evaluation there exists a function `psel` (short for: **p**reference **sel**ection) having the signature `psel(df, pref, ...)`. There, `df` is *data frame*, which is the most common way in R to represent data sets. The argument `pref` is some instance of a subclass of preference. With the three dots ("`...`") in the function signature some additional parameters can be specified for top-k selections. We will not go into the details here and refer to the rPref documentation. In Section 6.3.3 we will use some of the top-k functionality. Subsequently, we give a first example of the preference selection in rPref and show how data frames are constructed.

Example 6.2.1. Let A, B, C be atomic types with the domain $D_A = D_B = D_C = \mathbb{N}$. Assume a data set $r :: A \bowtie B \bowtie C$ given by:

$$r = 1 \bowtie 2 \bowtie 3 \;+\; 2 \bowtie 2 \bowtie 3 \;+\; 3 \bowtie 3 \bowtie 4 \;+\; 4 \bowtie 1 \bowtie 4 \;,$$

where $x :: A, y :: B, z :: C$ in every term $x \bowtie y \bowtie z$.

In R we create a data frame where each typename becomes a column name. The data frame constructor expects the data columnwise, e.g., the column of type A consists of the values $\{1, 2, 3, 4\}$. The `c` function is simply the concatenation operator in R.

```
r <- data.frame(A = c(1, 2, 3, 4), B = c(2, 2, 3, 1),
                C = c(3, 3, 4, 4))
```

Now we specify the preference $a = \mathsf{low}(A) \otimes \mathsf{high}(B) \otimes \mathsf{true}(C = 3)$ in rPref and calculate the maximum $s = a \rhd r$:

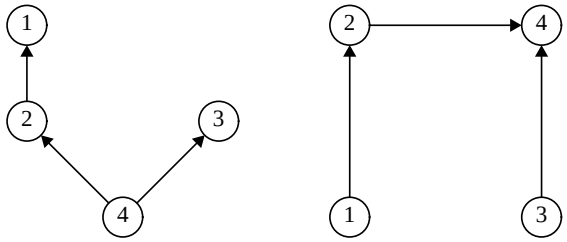
```
a <- low(A) * high(B) * true(C == 3)
s <- psel(r, a)
```

The first line of the code constructs a preference object, the second line evaluates the terms `A`, `B` and `C == 3` over the data set `r` and then calculates the Pareto optima. The result is again a data frame, having the same columns as the input. Subsequently, we show the result of this preference selection. Lines beginning with "`>`" indicate the R console input and the lines below are the corresponding output.

```
> s
  A B C
1 1 2 3
3 3 3 4
```

To plot the Hasse diagram of this preference for all cars in the data set, we can use `plot_btg(r, a, flip.edges = TRUE)` resulting in the graph in Figure 6.2 (left). Substituting `a` by `-a` returns the graph depicted in Figure 6.2 (right). The additional argument `flip.edges = TRUE` in the `plot_btg` function from rPref ensures that the edges are directed from worse to better elements w.r.t. the preference. Because of historical reasons, the default orientation of the edges is the other way around in rPref.

The labels of the vertices are the row numbers in the data set by default. In some cases the row numbers are not very meaningful. Hence there are more options to specify the labels, which is done, for example, in Figure 5.1 on page 57.



1) a 2) a^{-1}

Figure 6.2: The Hasse diagram for the preference from Example 6.2.1 and its converse relation, visualized in rPref.

In rPref, the *igraph* package [CN06] is used to plot graphs. Unfortunately, in that package there is no layouting devoted to Hasse diagrams of strict orders. Currently the layouting is done with the Reingold-Tilford algorithm [RT81], which is actually intended for trees. This implies that the vertices in some graphs are not sorted topologically. For example, the visualization of the preference a^{-1} in Figure 6.2(2), the edge from 2 to 4 is drawn horizontally instead of being oriented upwards (the node 4 should be on the top of the diagram). Nevertheless, this visualization functionality has been fully sufficient for our research. For example, when working on the decomposition algorithms, this graph visualization has turned out to be very helpful. With RGraphviz [GGH15] there is an R package better suited for layouting strict orders, but unfortunately this is not available on CRAN but only on the alternative package repository Bioconducter.

In base preferences like `true(C == 3)` the predicate `C == 3` is handled as an expression in R. For the constructor it does not matter if there exists some data set where the free variables in this term (here just `C`) are column names. Just when `psel(df, pref)` is called, then all expressions contained in the base preferences within `pref` are evaluated over the data set `df` and the preference is evaluated.

6.3 User Defined Preference Constructs in rPref

In rPref there are also constructors `low_`, `high_`, `true_` expecting R expressions as arguments. The term `low(C == 3)` is equivalent to `low_(expression(C == 3))`. This functionality can be used for building user defined preference constructors. Such functions returning a preference object and having a signature analogous to a base preference constructor are also called *base preference macro*. In the following we will describe some of them.

6.3.1 Base Preference Macros

Consider the logical opposite of the is_true preference, which is formally given by

$$\text{is.false}(\phi) = \text{is\_true}(\phi)^{-1} \ .$$

This can be specified in rPref by

```
false <- function(x, ...) -true_(substitute(x), ...)
```

The special R command `substitute(x)` causes the argument `x` not be evaluated but to be converted to an R expression. The minus operator is the reverse preference constructor. The three dots "`...`" bypass potential additional arguments for base preferences. The result of `false(C == 1)` is logically equivalent to `-true(C == 1)`. This is also the same as `true(C != 1)`.

For the preference decomposition from Chapter 5 the set preference $\mathtt{t}(p)$ plays an important role. For a data set $r = x_1 + ... + x_n$, we assume that there exists a column `id` which is a unique index for the x_i. The `pos(expr, value)` base preference macro, similar to the constructor of the same name in [Kie02], is already contained in rPref. With this, tuples where the set given by `expr` contains the subset `value` are preferred. Alternatively, it can be specified manually by:

```
pos <- function(expr, pos_value, ...) {
  true_(call('%in%', substitute(expr), pos_value), ...)
}
```

Again, potential additional arguments are bypassed via "`...`". The R function `call` constructs functions calls, where the first argument is the operator and the following arguments are the operands. This call formally constructs the logical predicate $(x \in p)$ where `expr` corresponds to x and `pos_value` to p. Based on this, we can formally define the set/tuple preference $\mathtt{t}(\cdot)$ by

```
t <- function(...) pos(id, c(...))
```

There the command `c(...)` computes the union of an arbitrary number of arguments of `t`, given by "`...`". Assume that the data set $r = x_1 + ... + x_n$ is represented in R by a data frame, given by `r <- data.frame(id = 1:n)`. We can define a tuple preference on x_1 by `t(1)` and evaluate it with `psel(r, t(1))`. A set preference $\mathtt{t}(x_2 + x_3)$ can be encoded as `t(2, 3)`. For example, we get the following output on the R console:

```
> t(2, 3)
[Preference] true(id %in% c(2, 3))
```

In rPref there are also some base preference macros, which internally call the constructor low. According to the framework described in [Kie02] there is an around($expr, val$) constructor which defines the wish, that the value of $expr$ and val should have a small absolute distance to each other. Formally, this corresponds to a $\mathsf{low}(\phi_v)$ preference with a distance function $\phi_v(x) = |x - v|$. In rPref, the `around` constructor effectively does the same. It generates a `low` preference with the help of the R built-in function `abs` calculating the absolute value. Consider the following console output:

```
> around(a, 5)
[Preference] low(abs(a - 5))
```

As another example, showing how new preference constructs are encoded in rPref, we consider the layered constructor as defined in [Kie05]. For an atomic type A it is formally given by

$$\text{layered}(A, L_0, ..., L_n) = \mathsf{low}(f_{L_0,..,L_n}) \quad \text{with} \quad f_{L_0,..,L_n} : D_A \to \mathbb{N},\ x \mapsto i \ \text{ for } \ x \in L_i \ .$$

The layer numbers coincide with the measure function for layered preferences, cf. Corollary 3.2.3. Using the construction from Lemma 5.1.9 which decomposes layered preferences into &-chains of Boolean preferences, this is equivalent to:

$$\text{layered}(A, L_0, ..., L_n) = \mathsf{is\_true}(A \in L_0) \ \& \ ... \ \& \ \mathsf{is\_true}(A \in L_n) \ .$$

Assuming that the layers are given as a list in R, we can encode the `layered_list` preference in rPref by:

```
layered_list <- function(expr, layers, ...) {
  expr <- substitute(expr)
  prefs <- lapply(layers,
                  function(x) true_(call('%in%', expr, x), ...))
  return(Reduce('&', prefs, empty()))
}
```

There, the attribute `expr` is converted to an R expression via `substitute` and will be later evaluated on a data set. The second line of the function performs the formal transformation from L_i to $\mathsf{is\_true}(A \in L_i)$ for every layer L_i. The built-in R function `Reduce` folds a list with a given operator, here the prioritisation operator &. Its third argument `empty()` corresponds to 0_T and is the neutral element for &. It acts as the initialization of the folding process and ensures that the `layered_list` function also works for empty lists.

We exemplify this for a numerical domain $D_A = \mathbb{N}$ and layers $L_0 = \{1, 2\}, L_1 = \{3\}$.

```
> layered_list(A, list(c(1, 2), 3))
[Preference] true(A %in% c(1, 2)) & true(A %in% 3)
```

In rPref there exists a `layered` constructor, which is implemented in a similar way, but expects the layers directly as arguments, i.e., `layered(A, L0, ..., Ln)`. This requires some more special R techniques, hence we refrain from explaining it here.

Such user defined preference constructors are also used in the implementation of preferences on hierarchies, cf. [RWR+15]. There, constructors are defined which search for objects having a best matching category, where the category hierarchy is given in a data set. We will not go into details here.

In the next section we will consider another example from a textual domain.

6.3.2 Preferences for Regular Expression Matching

In [RK13] we used the rapid prototyping system R-Pref for a text-mining use case. These textual preferences, using the regular expression functionality from R, have been encoded in a quite lengthy way and in an imperative style in R-Pref.

Using the rPref package we can realise text matching preferences much more concisely and in a functional style. There is a built-in R function `grepl(regex, txt)` returning a logical `true` value if the regular expression `regex` is found in the string `txt` and `false` otherwise. This function works also *vectorwise*, i.e., if `text` is a *vector* (the concept for arrays in R) of strings, then a vector of logical values is returned. Hence we can define a preference `true(grepl(regex, A))` for a regular expression `regex` and a text column `A`. In

the following example we present a use case showing how complex preference operators and regular expression matching can be combined.

Example 6.3.1. Let A, B atomic types with $D_A = \mathbb{N}$ and $D_B = \{\text{'a'}, \text{'b'}\}^*$. This means, D_B contains all finite sequences of a's and b's. Assume an example data set

$$r = 1 \bowtie \text{'a'} + 2 \bowtie \text{'ab'} + 3 \bowtie \text{'ba'} + 4 \bowtie \text{'aaa'} + 5 \bowtie \text{'aaba'} ,$$

which is encoded in R via

```
r <- data.frame(id = 1:5,
                text = c("a", "ab", "ba", "aaa", "aaba"))
```

We consider the following preferences:

- Using the Boolean preference `true(grepl("aa|ba", text))`, we get the following three tuples:

```
> psel(r, true(grepl("aa|ba", text)))
id text
3  3   ba
4  4  aaa
5  5 aaba
```

- Taking the Pareto preference `true(grepl("aa", text)) * true(grepl("ba", text)`, just the tuple with id 5 is returned, as both sequences 'aa' and 'ba' are contained only in the string 'aaba'. But the tuples 3 and 4 are the next best alternatives, as we see from the Hasse diagram in Figure 6.3.

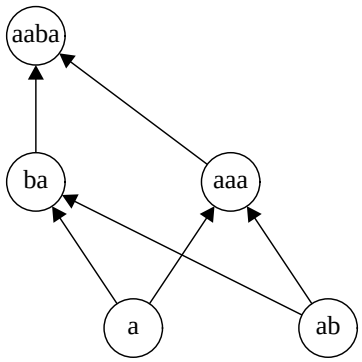


Figure 6.3: The Hasse diagram for the preference from Example 6.3.1 visualized in rPref.

Hence by substituting the or-operator `|` in a regular expression by a Pareto operator on the level of preferences we can refine the order: all tuples containing 'aa' or 'ba' are still better than those, which do not contain these sequences. But a tuple containing both 'aa' and 'ba' dominates all tuples, which contain at most one of the sequences.

6.3.3 Implementation of the Layered Preference Transformation

In this section we will show how the layered preference transformation from Section 3.5 can be implemented with rPref.

First, we revisit the definition of layers. According to Definition 3.5.3 the layer-i elements for a preference $a :: T^2$ and a data set $r :: T$ are given by

$$q_i =_{df} a \rhd r_i \quad \text{where} \quad r_i =_{df} r - \sum_{j=0}^{i-1} q_j \quad \text{for} \quad i = 0, 1, 2, \ldots .$$

This layer number can be retrieved in rPref using the top-k interface of the preference evaluation function `psel`. A query `psel(r, a, top = k)` returns exactly k tuples. If $|q_0| > k$, i.e., the first layer $q_0 = a \rhd r$ contains more tuples than requested, some tuples from the result are non-deterministically removed. If $|q_0| < k$, i.e., more tuples as contained in the first layer are requested, then additional tuples have to be added. There tuples are taken iteratively from the layers in the sequence $(q_i)_i$, until we have k tuples. By using `top = n` for a data set with n tuples, we will get tuples from all layers, i.e., the entire data set is returned. Additionally `psel` returns the layer number i of every tuple if we additionally specify the optional argument `show_level = TRUE`. Hence we can calculate all the q_i layers with the top-k interface of rPref. We will subsequently exemplify this.

Example 6.3.2. Assume the data set r and preference a as given in Example 6.2.1, i.e., we have

$$r = 1 \bowtie 2 \bowtie 3 \;+\; 2 \bowtie 2 \bowtie 3 \;+\; 3 \bowtie 3 \bowtie 4 \;+\; 4 \bowtie 1 \bowtie 4, \quad a = \mathsf{low}(A) \otimes \mathsf{high}(B) \otimes \mathsf{true}(C = 3) \; .$$

Now, in rPref we call the top-k selection for $k = n$, where n is the number of tuples in the data set `r` (in R obtained by `nrow(r)`) and show the level numbers:

```
> psel(r, a, top = nrow(r), show_level = TRUE)
  A B C .level
1 1 2 3      1
3 3 3 4      1
2 2 2 3      2
4 4 1 4      3
```

The special column `.level` is appended because `show_level = TRUE` is given as additional parameter. This column corresponds to the layer number, beginning at 1 (and not at 0, as it is the case for the formal definition of the q_i). From this result we obtain the layers

$$q_0 = 1 \bowtie 2 \bowtie 3 \;+\; 2 \bowtie 2 \bowtie 3 \; ,$$
$$q_1 = 3 \bowtie 3 \bowtie 4 \; ,$$
$$q_2 = 4 \bowtie 1 \bowtie 4 \; .$$

Next, we will encode the induced layered preference from Definition 3.5.6 based on the top-k preference selection in rPref. According to the definition, tuples with greater layer numbers are better than tuples with smaller layer numbers. Tuples from the same layer are in one equivalence class. Hence we can express the $m(a, r)$ preference for $a :: T^2$ and a data set $r :: T$ formally by

$$m(a, r) = \mathsf{low}(f_r) \quad \text{with} \quad f_r : D_T \to \mathbb{N},\; x \mapsto i \;\text{ for }\; x \in q_i \; ,$$

where the q_i are defined as above. The parameter r is implicitly used in the function f_r for calculating the layers q_i. This can be translated to an R function `m(a, r)`, based on rPref, as shown subsequently.

```
m <- function(a, r) {
  df <- psel(r, a, top = nrow(r), show_level = TRUE)
  low(df[row.names(r), '.level'], df = df)
}
```

First, the level values for each tuple are calculated and stored in `df[['.level']]`. Next, the mapping of each tuple index from to its layer number is constructed by the succeeding code line. Therefore every tuple index, in R the `row.names` of the data frame `r`, is mapped to the `.level` value in the preference selection result `df`. The additional argument `df = df` in the `low` constructor causes an immediate evaluation of this mapping on the given data frame, which is done for a better readability of the resulting preference object. We have a look at the result:

```
> m(a, r)
[Preference] low(c(1, 2, 1, 3))
> psel(r, m(a,r), top = nrow(r))
  A B C .level
1 1 2 3      1
3 3 3 4      1
2 2 2 3      2
4 4 1 4      3
```

Note that the `psel` function above returns a static vector of level numbers. The base preference `low` interprets it as a mapping ϕ from the tuple indices (i.e., `row.names` in the data set `r`) to the values contained in this static vector. Hence $\mathsf{low}(\phi)$ prefers those tuples having the lowest layer numbers, in this case the tuples with indices 1 and 3, which are both in the first layer. The result of `psel(...)`, which is given above, shows that we have reconstructed the original preference `a`.

But such a preference is constant w.r.t. the data set. Calling `psel(s, m(a, r))` with some data set `s` not being identical to `r` would result in undefined behaviour. Because of this, we will sketch a more elegant approach for implementing the $m(a, r)$ transformation.

In rPref, there exists a special variable `df__` which can be used in the argument of any base preference `low`, `high` and `true`. It offers the access the entire data set given in a call of `psel`. When the preference evaluation is done by calling `psel(r, a)`, the variable `df__` occurring in the definition of `a` is substituted by `r`. Hence we can define the induced layered preference without explicitly stating the data set but using its abstract representation `df__`. This leads to the following function `m_(a)`.

```
m_ <- function(a) {
  low({ df <- psel(df__, a, top = nrow(df__), show_level = TRUE)
        df[row.names(df__), '.level'] })
}
```

Now when `psel(r, m_(a))` is called and internally `df__ <- r` is assigned, the above definition of `low` becomes equivalent to that of `m(a, r)`. In `m_` the calculation `df <- psel(...)` must be done within the argument of the `low` constructor, as `df__` is available only when the resulting preference of `m_` is processed within `psel`. The mapping of tuples to their levels within `m_` depends on the data set `r` given in `psel(r, m_(a))`. Its result is equivalent to `psel(r, m(a, r))`.

Based on this `m_` function we can finally implement the regularised prioritisation as given in Definition 3.5.9. For preferences a and b it is defined by $a \mathbin{\&_{\mathrm{reg}}} b = m(a \bowtie \top_b, r) \mathbin{\&} b$ and implicitly depends on the data set r. The `m_` function from above allows encoding the $\&_{\mathrm{reg}}$ operator independently of the data set. This is done with the following code, where a function beginning and ending with `%` is a user-defined infix operator in R.

```
"%&reg%" <- function(a, b) {
  m_(a) & b
}
```

In the definition of an infix operator in R, quotations marks have to be put around the `%` chars. For preference objects `a` and `b` this operator is called by `a %&reg% b`.

Now we revisit Example 3.5.11 and comprehend its results in R.

Example 6.3.3. Assume the data set $r = 1 \bowtie 2 \bowtie 1 + 2 \bowtie 1 \bowtie 2 + 2 \bowtie 2 \bowtie 3$ and the preferences a, b, as given in Example 3.5.11,

$$a = (\mathsf{low}(A) \otimes \mathsf{low}(B)) \,\&\, \mathsf{low}(C) \ ,$$
$$b = (\mathsf{low}(A) \otimes \mathsf{low}(B)) \,\&_{\mathrm{reg}}\, \mathsf{low}(C) \ .$$

In rPref the data set and the preference selections are encoded with:

```
r <- data.frame(A = c(1, 2, 2), B = c(2, 1, 2), C = c(1, 2, 3))
a <- (low(A) * low(B)) & low(C)
b <- (low(A) * low(B)) %&reg% low(C)
```

Now we retrieve the following results:

```
> psel(r, a)
  A B C
1 1 2 1
2 2 1 2

> psel(r, b)
  A B C
1 1 2 1
```

This is in accordance with the results of Example 3.5.11. While the usual prioritisation used in a preserves the incomparability of the tuples $1 \bowtie 2 \bowtie 1$ and $2 \bowtie 1 \bowtie 2$, the regularised prioritisation allows refining this result and just returns $1 \bowtie 2 \bowtie 1$.

CHAPTER 7

Conclusion and Outlook

In this thesis we discussed various theoretical aspects of database preferences. We presented a point-free relational formalization of preferences and an algebraic approach based on the join algebra. We showed many results playing an important role for the practical aspects of our work, e.g., the preference decomposition. This was followed by a presentation of case studies, where chosen theoretical constructs are implemented using our rPref package. Now, we will summarize our work and point out some questions for future research.

7.1 Summary

We formalized the preference framework as presented in [Kie02] and [Kie05] in an algebraic way. Many of the proofs based on point-wise arguments in the original publications could be abbreviated using the algebraic approach. Moreover, we precisely axiomatized the preference framework by means of the join algebra. By doing this we could see which additional requirements, compared to the abstract relation algebra, are needed for specifying the preference operators.

As a non-trivial example we picked the distributive law for & and ⊗ and showed this theorem using Prover 9. We could axiomatize the typing mechanism and the join algebra in the Prover, but we did not achieve a fully automated one-step proof. The underlying structures are very complex in this case, resulting in a search tree too large for the prover.

Using the algebraic methods we could investigate practical aspects of preferences, as in Chapter 4. There we proved different kinds of optimizations, which are relevant for implementing a preference interpreter. These results found there are not new, but the kind of proofs is new, as many arguments directly rely on the algebraic properties of the underlying structures like the abstract relation algebra and join algebra. Thus the proofs became shorter and better readable, compared to a pure relational and point-wise approach.

The rPref package is the most application oriented part of our work. It was developed during the last two years of working on this thesis. It implements the described preference framework while sticking close to its formal foundations. For our research it was important in two different regards. First, for the research concerning the preference decomposition the package served as an important tool to evaluate our algorithms at an early stage. Second, preference constructs like the regularization were implemented in a very similar fashion as

in the theoretical parts. Furthermore, most of the given examples and algorithms in this thesis can be easily reproduced using that package and the source code snippets available in the appendix and on the web.

7.2 Future Work

In some of the application related parts of this thesis, there are still relational and point-wise arguments, e.g., the proofs of the decomposition theorems in Section 5.2. It is an open question if there exist algebraic structures which allow shortening these proofs.

Some of the point-wise proofs from [KH03] regarding grouped preferences could be simplified as shown in Section 4.1. For other problems in that paper this approach was less successful. For example, there are optimization laws for preference queries in connection with conditional joins between different data tables. These arguments seem to be inherently point-wise and it is an open question if there exists a smart way to transform the proofs of these laws to the algebraic setting.

Related to the preference decomposition from Chapter 5, the question remains on how to find minimal representations of preferences. We have shown that the complexity of one of the decomposition methods is very high. The other approach, using just the Pareto operator, generates preference terms which are restricted by the number of tuples in the data set. Such preferences also induce an n-dimensional lattice, cf. [PK07]. Clearly, our algorithms do not provide the most compact possible representation of preferences. In [EP16] the authors present an algorithm for lattice-based representations of preferences, which are more compact than in our approach but not minimal. Finding the shortest possible number of Pareto operators connecting layered preferences, is an interesting question for future research. Such kinds of representation could potentially also lead to a more efficient processing of preference queries.

The use of an automated theorem prover, exemplified for the distributive law, reached its limitations for complex structures like the join algebra. It is an open question, how such proofs can be further automatized. Alternatively, interactive theorem provers like *Isabelle* could be used for showing the correctness of preference optimization laws and algorithms. We briefly evaluated the use of Isabelle for the join algebra, but perceived the complexity of these systems as disproportionately high.

Seen from the perspective of a database user, a high-performance implementation of database preferences is available by means of the commercial database system Exasolution. The preference language of that is very similar to the theoretical framework described in this thesis. For future research and development, powerful algorithms for a large-scale preference processing are of high interest. The pioneering paper about the Skyline operator [BKS01] was published about 15 years ago, but there are still few applications in the big data context. One potential reason for this is that retrieving maximal elements w.r.t. a preference causes quadratic costs in the worst case and hence is limited scalable. Another potential drawback is, that there are many different approaches for Skyline processing.

The approach of our freely available rPref package is to provide a reliable and expandable preference interpreter. New preference constructs can be easily evaluated, as we have shown in several parts of this thesis. Hence we think, that the described preference framework in connection with the available tools is a promising candidate for future research and new developments in the area of database preferences.

Appendix A

Code Examples

A.1 Model Finder: Unique Tuple Decompositions

Given the data set $r = x_1 + ... + x_4$ and the preference

$$a = ((\mathsf{t}(x_1) \otimes \mathsf{t}(x_3)) \,\&\, \mathsf{t}(x_2)) \otimes (\mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4)) \ ,$$

there is no decomposition into an r-equivalent preference within $\mathsf{un}_{\{\&,\otimes\}}(r)$. We will show this in the following R script.

The model finder function `search` extends the temporary preference term `a_tmp` by $... \otimes \mathsf{t}(x_i)$ or $... \,\&\, \mathsf{t}(x_i)$ in the recursive step. The term extension at the end is sufficient to get all possible terms, as & and $\otimes$ are associative operators. The variable `xs` stores those $x \in r$ which can still be used for $\mathsf{t}(x)$ without violating the uniqueness of the x_i. Hence `xs` is comparable to the parameter s in $\mathsf{un}_{\mathsf{op}}(s)$, Definition 5.1.2.

The comparison w.r.t. r-equivalence of two preferences (cf. Definition 5.1.3) is done by comparing the adjacency lists of their Hasse diagrams. Note that we can rely on a predefined sorting (lexicographic) of the adjacency list of a Hasse diagram in the result of `get_hasse_diag`. Hence the equivalence of these adjacency lists imply the r-equivalence of the corresponding preferences.

```
# include the rPref package
library(rPref)

# define set preference
t <- function(...) pos(id, c(...))

# implement an r-equality check of a and b
# (the Hasse diagram is uniquely sorted)
is_r_equal <- function(a, b, r)
  identical(get_hasse_diag(r, a), get_hasse_diag(r, b))

# main procedure for the model finder
do_search <- function(r, b) {

  # define recursive search for unique tuple decompositions
  search <- function(a_tmp, xs) {

    # check if temporary preference is equivalent to b
    if (is_r_equal(a_tmp, b, r)) {
      # print preference term on console and return TRUE
      print(a_tmp)
      return(TRUE)
    }

    # recursively search for other possible terms
```

```
    if (length(xs) > 0) {
      for (x in xs) {
        if (search(a_tmp & t(x), setdiff(xs, x))) return(TRUE)
        if (search(a_tmp * t(x), setdiff(xs, x))) return(TRUE)
      }
    }

    # return FALSE if no path returned TRUE
    return(FALSE)
  }

  # start recursive search
  return(search(empty(), r[['id']]))
}

# data set and given preference to decompose a_ref
r <- data.frame(id = 1:4)
a_ref <- ((t(1) * t(3)) & t(2)) * (t(3) & t(4))
```

Finally, we consider the return value of the `do_search` function:

```
> do_search(r, a_ref)
[1] FALSE
```

Here the value `FALSE` means that no decomposition is found. This script generates and checks 633 possible terms. On our off-the-shelf computer the execution time of this program is about 8 seconds.

In contrast to that consider the total order $a = \mathsf{t}(x_1) \,\&\, \mathsf{t}(x_2) \,\&\, \mathsf{t}(x_3) \,\&\, \mathsf{t}(x_4)$, where a preference term is found within $\mathsf{un}_{\{\&,\otimes\}}(r)$:

```
> do_search(r, t(1) & t(2) & t(3) & t(4))
[Preference] true(id %in% 1) & true(id %in% 2) & true(id %in% 3)
[1] TRUE
```

The term $\mathsf{t}(x_1) \,\&\, \mathsf{t}(x_2) \,\&\, \mathsf{t}(x_3)$ is returned which is clearly r-equivalent to a on the data set $r = x_1 + \ldots + x_4$.

Note that the `search` function here contains some kind of a brute-force search. It could be optimized by e.g., exploiting the commutativity of $\otimes$ and a more efficient r-equivalence check. We omitted such optimizations to keep the code as simple as possible.

A.2 Decomposition Algorithms in rPref

Subsequently, we show the source code, based on rPref, for the decomposition algorithms DECOMP_PARETO(a, r) and DECOMP_TUPLE(a, r) from Section 5.2.

```
# include the rPref package
library(rPref)

# set preference for multiple arguments
# t(1, 2) corresponds to t(x1 + x2) (formally)
t <- function(...) pos(id, c(...))

# Decomp_Pareto algorithm from Theorem 5.2.1
decomp_pareto <- function(a, r) {

  # initialize predecessors and successors of the Hasse diagram of a
  # (this has to be called in rPref before calling pred and succ)
  init_pred_succ(r, a)
```

```
  # compose preferences; 'all_pred(a, x)' corresponds to '<a|x'
  b <- lapply(1:nrow(r), function(x) t(c(x, all_pred(a, x))))

  # generate Pareto composition
  return(Reduce('*', b, empty()))
}

# Decomp_Tuple algorithm (Algorithm 5.1, Theorem 5.2.4)
decomp_tuple <- function(a, r) {

  # select maxima, corresponds to 'a |> r'
  m <- psel(r, a)$id

  # initial value of array b (b is implemented as an R list)
  b <- list()
  b[1:nrow(r)] <- empty() # empty preference, corresponding to 0_a

  # initialize predecessors and successors of the Hasse diagram of a
  init_pred_succ(r, a)

  # traverse the Hasse diagram
  while(length(m) > 0) {

    # implementation of the for-loop in an lapply-function
    b[m] <- lapply(m, function(x) {
      # bx corresponds to 'b[x] with x \in r * <a_h|y' in line 7
      bx <- b[hasse_pred(a, x)]
      # Pareto-combine the bx add sub-prioritized t(x)
      return(Reduce('*', bx, empty()) & t(x))
    })

    # corresponds to 'b[r * <a_h|m] = 0'
    b[hasse_pred(a, m)] <- empty()

    # successors for next step, 'a |> (r * |a_h>m)'
    m <- psel(r[hasse_succ(a, m),,drop=FALSE], a)$id
  }

  # Pareto-compose all b[x] and return
  return(Reduce('*', b, empty()))
}

# example runs corresponding to Example 5.1.10

# data set x1 + ... + x5
r <- data.frame(id = 1:5)

# transform tuple decomposition to Pareto decomposition
a <- (((t(1) & t(3)) * t(2)) & t(4)) * t(5)
a_ <- decomp_pareto(a, r)

# transform Pareto decomposition into tuple decomposition
b <- t(1) * t(2) * t(1, 3) * t(1, 2, 3, 4) * t(5)
b_ <- decomp_tuple(b, r)
```

Subsequently, we show the results of both decompositions from above.

```
> a_
[Preference] true(id %in% 1) * true(id %in% 2) *
  true(id %in% c(3, 1)) * true(id %in% c(4, 1, 2, 3)) *
  true(id %in% 5)
> b_
[Preference] ( ( true(id %in% 2) *
  (true(id %in% 1) & true(id %in% 3)) ) & true(id %in% 4) ) *
  true(id %in% 5)
```

These results correspond to a' and b' in Example 5.1.10. We added some line breaks and whitespaces to the console output for sake of readability.

A.3 Optimized Decomposition Algorithms in rPref

We give the source code, based on rPref, for the decomposition algorithms $\text{DEC\_MIN}_1(a, r)$ and $\text{DEC\_MIN}_2(a, r)$ from Section 5.3. As some parts of the code are very similar to the standard decomposition algorithms from the previous section, we do not comment the analogous parts of the code.

```
# include rPref and dplyr packages, define set preference
library(rPref)
library(dplyr) # used for constructing the minimized preference
t <- function(...) pos(id, c(...))

# construct minimized preference
minimize_pref <- function(a, r) {

  # initialize hasse_pred/hasse_succ function
  init_pred_succ(r, a)

  # calculate predecessors/successors and serialize them
  eq_table <- data.frame(node_id = r$id, predsucc = vapply(r$id,
      function(x) {
        paste0(paste(hasse_pred(a, x), collapse = ","), ";",
               paste(hasse_succ(a, x), collapse = ","))
      }, ""))

  # get equivalence classes by a "group by" operation
  min_df <- as.data.frame(summarise(group_by(eq_table, predsucc),
                          id = min(node_id), eq_ids = list(node_id)))

  # add row numbers (for hasse_pred/hasse_succ functions) and return
  min_df[['row']] <- 1:nrow(min_df)
  return(min_df[,c('row','id','eq_ids')])
}

# decomposition into Pareto preferences with minimization
dec_min_1 <- function(a, r) {

  # generate table of equivalence classes
  r_min <- minimize_pref(a, r)

  # set preference for equivalence classes
  t_min <- function(...)
    # transform row ids to equivalence class ids
    pos(id, do.call('c', subset(r_min, row %in% c(...))$eq_ids))

  # same code as decomp_pareto with r -> r_min, id -> row, t -> t_min
  init_pred_succ(r_min, a)
  b <- lapply(r_min$row, function(x) t_min(x, all_pred(a, x)))
  return(Reduce('*', b, empty()))
}

# decomposition into {&, *} with minimization
dec_min_2 <- function(a, r) {

  # generate table of equivalence classes
  r_min <- minimize_pref(a, r)

  # set preference for equivalence classes
  t_min <- function(...)
    pos(id, do.call('c', subset(r_min, row %in% c(...))$eq_ids))
```

```
  # same code as decomp_tuple with r->r_min, id->row, t->t_min
  m <- psel(r_min, a)$row
  b <- list()
  b[r_min$row] <- empty()
  init_pred_succ(r_min, a)
  while(length(m) > 0) {
    b[m] <- lapply(m, function(x) {
      b_x <- b[hasse_pred(a, x)]
      return(Reduce('*', b_x, empty()) & t_min(x))
    })
    b[hasse_pred(a, m)] <- empty()
    m <- psel(r_min[hasse_succ(a, m),], a)$row
  }
  return(Reduce('*', b, empty()))
}

# example runs corresponding to Example 5.3.1

# given data set and preference
r <- data.frame(id = 1:5)
a <- t(1, 2)

# optimized decompositions
a1 <- dec_min_1(a, r)
a2 <- dec_min_2(a, r)
```

Subsequently, we show the results of both decompositions from above.

```
> a1
[Preference] true(id %in% 1:2) * true(id %in% 1:5)
> a2
[Preference] true(id %in% 1:2) & true(id %in% 3:5)
```

These results correspond to those in Example 5.3.1. Note that the extended versions of these algorithms available at [Roo15b] also support the output of equivalence classes within the preference terms, e.g., $\mathsf{t}([\![x_1]\!])$ instead of $\mathsf{t}(x_1 + x_2)$.

A.4 Triangle Preference in rPref

Subsequently, we show the R code to construct the triangle preference from Definition 5.5.2.

```
# rPref package and set preference
library(rPref)
t <- function(...) pos(id, c(...))

# create the data set for the triangle preference
r_ <- function(k) {
  data.frame(id = 1:(k * (k + 1) / 2))
}

# function to create the triangle preference
a_ <- function(k) {

  # data set and the number of nodes
  r <- r_(k)
  n <- nrow(r)

  # initialize preference list for all nodes
  b <- list()
  b[1:n] <- empty()
```

```
  # first row
  b[1:k] = lapply(1:k, t)

  # initialization for row 1 (tuples 1, 2, ..., k)
  ind <- k
  row_inds <- 1:k

  # cycle through rows 2, 3, ..., (k-1)
  if (k >= 3) {
    for (i in 2:(k-1)) {

      last_row_inds <- row_inds
      row_inds <- (row_inds + (k+1-i))[-1]

      # cycle through columns
      for (j in 1:(k+1-i)) {
        ind <- ind + 1
        # predecessors
        pred <- setdiff(last_row_inds, ind - (k+2-i))
        # preference for current node
        b[[ind]] <- Reduce('*', b[pred], empty()) & t(ind)
      }
    }
  }

  # last row k, compose final preference and return
  return((b[[row_inds[1]]] * b[[row_inds[2]]]) & t(n))
}
```

Now we show the triangle preference for `k = 3` in the console (with additional white-spaces for readability) and plot the Hasse diagram of this preference (or Better-Than-Graph, short BTG).

```
> a_(3)
[Preference]
  ( ((true(id %in% 2) * true(id %in% 3)) & true(id %in% 4)) *
    ((true(id %in% 1) * true(id %in% 3)) & true(id %in% 5))   ) &
  true(id %in% 6)
> plot_btg(r_(3), a_(3), flip.edges = TRUE)
```

The term `a_(3)` corresponds to DECOMP_TUPLE$(a(3), r(3))$ where $a(\cdot)$ and $r(\cdot)$ are given as in Definition 5.5.2. In Figure A.1 we show the graphical output of the `plot_btg` function.

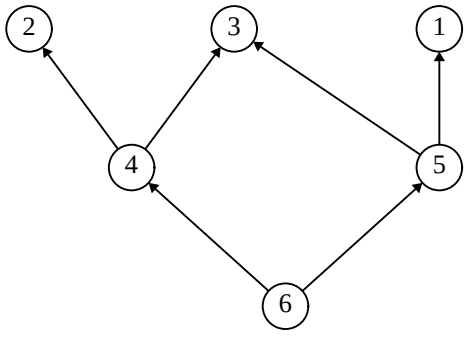


Figure A.1: Triangle preference for $k = 3$.

Bibliography

[BKS01] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *17th International Conference on Data Engineering*, pages 421–430, 2001.

[BR01] C. Brink and I. Rewitsky. *A Paradigm for Program Semantics: Power Structures and Duality.* CSLI Publications, Stanford, California, 2001.

[Cho03] J. Chomicki. Preference Formulas in Relational Queries. In *TODS '03: ACM Transactions on Database Systems*, volume 28, pages 427–466, New York, NY, USA, 2003.

[CN06] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[DD95] H. Darwen and C. J. Date. The Third Manifesto. *ACM SIGMOD Record*, 24(1):39–49, 1995.

[DGM+14] H.-H. Dang, R. Glück, B. Möller, P. Roocks, and A. Zelend. Exploring Modal Worlds. *Journal of Logical and Algebraic Methods in Programming*, 83(2):135–153, 2014.

[DMS06] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Trans. Comput. Logic*, 7(4):798–833, October 2006.

[Edd13] D. Eddelbuettel. *Seamless R and C++ integration with Rcpp.* Springer, 2013.

[Eil74] S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, 1974.

[End11] M. Endres. *Semi-Skylines und Skyline Snippets - Theory and Applications.* University of Augsburg, Dissertation. Books on Demand GmbH, Norderstedt, 2011.

[EP16] Markus Endres and Timotheus Preisinger. Preference Structures and their Lattice Representations. Technical Report 2016-02, University of Augsburg, 2016.

[ERK14] M. Endres, P. Roocks, and W. Kießling. Algebraic optimization of grouped preference queries. In *18th International Database Engineering & Applications Symposium*, pages 247–256. ACM, 2014.

[ERK15] M. Endres, P. Roocks, and W. Kießling. Scalagon: An Efficient Skyline Algorithm for All Seasons. In *DASFAA '15: Database Systems for Advanced Applications*, pages 292–308. Springer, 2015.

[ERW+12] M. Endres, P. Roocks, F. Wenzel, A. Huhn, and W. Kießling. Handling of NULL Values in Preference Database Queries. In *6th Multidisciplinary Workshop on Preference Handling (MPREF) in conjunction with AAA-12*, 2012.

[Exa15] Exasol. Skyline. In *EXASolution User Manual Version 5.0.0*, pages 239–241. 2015. `http://tinyurl.com/mob2mfm`.

[Fis70] P. C. Fishburn. Utility theory for decision making. Technical report, New York, USA, 1970.

[GGH15] Jeff Gentry, Robert Gentleman, and Wolfgang Huber. How To Plot A Graph Using Rgraphviz. 2015. `http://tinyurl.com/gtrsx6j`.

[Gis88] J. L. Gischer. The Equational Theory of Pomsets. *Theoretical Computer Science*, 61(2):199–224, 1988.

[HK05] B. Hafenrichter and W. Kießling. Optimization of Relational Preference Queries. In *ADC '05: 16th Australasian database conference*, pages 175–184, Darlinghurst, Australia, 2005.

[HS08] P. Höfner and G. Struth. On Automating the Calculus of Relations. In *Automated Reasoning*, pages 50–66. Springer, 2008.

[Kan89] P. C. Kanellakis. Elements of Relational Database Theory. Technical report, Providence, RI, USA, 1989.

[KEW11] W. Kießling, M. Endres, and F. Wenzel. The Preference SQL System – An Overview. *Bulletin of the Technical Commitee on Data Engineering, IEEE Computer Society*, 34(2):11–18, 2011.

[KH03] W. Kießling and B. Hafenrichter. Algebraic Optimization of Relational Preference Queries. Technical Report 2003-1, University of Augsburg, 2003.

[Kie02] W. Kießling. Foundations of Preferences in Database Systems. In *VLDB '02: 28th International Conference on Very Large Data Bases*, pages 311–322, Hong Kong, China, 2002. VLDB.

[Kie05] W. Kießling. Preference Queries with SV-Semantics. In *COMAD '05: Advances in Data Management 2005, 11th International Conference on Management of Data*, pages 15–26, Goa, India, 2005.

[KK02] Werner Kießling and Gerhard Köstler. Preference SQL: Design, Implementation, Experiences. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 990–1001. VLDB Endowment, 2002.

[KRR02] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB '02: 28th international conference on Very Large Data Bases*, pages 275–286. VLDB Endowment, 2002.

[LYL09] M.-H. Luk, M. L. Yiu, and E. Lo. Group-by Skyline Query Processing in Relational Engines. In *18th ACM conference on Information and knowledge management*, CIKM '09, pages 1433–1436. ACM, 2009.

[Mad97] R. D. Maddux. *Relation Algebras*, chapter 2, pages 22–38. Advances in Computing. Springer-Verlag, Wien, New York, 1997.

[MB85] E. G. Manes and D. B. Benson. The Inverse Semigroup of a Sum-ordered Semiring. *Semigroup Forum*, 31(1):129–152, 1985.

[McC05] W. McCune. *Prover9 and Mace4*, 2005. `https://www.cs.unm.edu/~mccune/prover9/`.

[MKEK15] S. Mandl, O. Kozachuk, M. Endres, and W. Kießling. Preference Analytics in EXASolution. 16th Conference on Database Systems for Business, Technology, and Web. 2015. `http://tinyurl.com/pxco8d4`.

[Möl15] B. Möller. Towards Antichain Algebra. In *RAMiCS '15: Relational and Algebraic Methods in Computer Science*, volume 9348 of *Lecture Notes in Computer Science*, pages 344–361. 2015.

[MO04] W. MacCaull and E. Orłowska. A Calculus of Typed Relations. In *Relational and Kleene-Algebraic Methods in Computer Science*, volume 3051 of *Lecture Notes in Computer Science*, pages 191–201. 2004.

[MPJ07] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains. In *VLDB '07: 33rd International Conference on Very Large Data Bases*, pages 267–278. VLDB Endowment, 2007.

[MR12a] B. Möller and P. Roocks. *Proof of the Distributive Law for Prioritisation and Pareto Composition*, 2012. `http://www.p-roocks.de/phd/distrib_2012.pdf`.

[MR12b] B. Möller and P. Roocks. An Algebra of Layered Complex Preferences. In *RAMiCS '12: Relational and Algebraic Methods in Computer Science*, volume 7560 of *Lecture Notes in Computer Science*, pages 294–309. 2012.

[MR15] B. Möller and P. Roocks. An Algebra of Database Preferences. *Journal of Logical and Algebraic Methods in Programming*, 84(3):456 – 481, 2015.

[MRE12] B. Möller, P. Roocks, and M. Endres. An Algebraic Calculus of Database Preferences. In *MPC '12: Mathematics of Program Construction*, volume 7342 of *Lecture Notes in Computer Science*, pages 241–262. 2012.

[PK07] T. Preisinger and W. Kießling. The Hexagon Algorithm for Evaluating Pareto Preference Queries. In *MPref '07: 3rd Multidisciplinary Workshop on Advances in Preference Handling*, 2007.

[R C15] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. `http://www.R-project.org`.

[REK13] P. Roocks, M. Endres, and W. Kießling. Specification and Optimization of Preference (SV-) Grouping Queries. Technical report, University of Augsburg, 2013.

[REMK12] P. Roocks, M. Endres, S. Mandl, and W. Kießling. Composition and Efficient Evaluation of Context-Aware Preference Queries. In *DASFAA '12: Database Systems for Advanced Applications*, 2012.

[RK13] P. Roocks and W. Kießling. R-Pref: Rapid Prototyping of Database Preference Queries in R. In *DATA '13: 2nd International Conference on Data Management Technologies and Applications*, pages 104–111, 2013.

[Roo12] P. Roocks. A Hierarchically Typed Relation Algebra. In *Student paper track of: Relational and Algebraic Methods in Computer Science*, 2012.

[Roo13] P. Roocks. *R-Pref Documentation and Sources*, 2013. `http://ursaminor.informatik.uni-augsburg.de/trac/wiki/R-Pref`.

[Roo15a] P. Roocks. Decomposition of Database Preferences on the Power Set of the Domain. In *RAMiCS '15: Relational and Algebraic Methods in Computer Science*, volume 9348 of *Lecture Notes in Computer Science*, pages 362–379. 2015.

[Roo15b] P. Roocks. *Examples and algorithms for the (power set) preference decomposition algorithms*, 2015. `http://www.p-roocks.de/phd/`.

[Roo15c] P. Roocks. Preference Decomposition and the Expressiveness of Preference Query Languages. In *MPC '15: Mathematics of Program Construction*, pages 71–92. 2015.

[Roo15d] P. Roocks. *Prover9 input files for Distributive Law for Prioritisation and Pareto Composition*, 2015. `http://www.p-roocks.de/phd/prover_input.zip`.

[Roo15e] P. Roocks. *R script for the Scalagon performance study*, 2015. `http://www.p-roocks.de/phd/alpha.r`.

[Roo15f] P. Roocks. *The rPref Package: Preferences and Skyline Computation in R*, 2015. `http://www.p-roocks.de/rpref`.

[RT81] E. M. Reingold and J. S. Tilford. Tidier Drawings of Trees. *IEEE Transactions on Software Engineering*, (2):223–228, 1981.

[RWR+15] P. Roocks, F. Wenzel, O. Rudenko, M. Endres, and W. Kießling. A Database Approach for Categorical Preferences on Hierarchies. In *9th Multidisciplinary Workshop on Preference Handling (MPREF)*, 2015.

[SS93] G. Schmidt and T. Ströhlein. *Relations and Graphs: Discrete Mathematics for Computer Scientists*. EATCS Monographs on Theoretical Computer Science, 1993.

[TPC15] TPC. *TPC Benchmark H*, 2015. `http://tpc.org/TPC_Documents_Current_Versions/pdf/tpch2.17.1.pdf`.

[WBKH04] Q. Wang, W.-T. Balke, W. Kießling, and A. Huhn. P-News: Deeply Personalized News Dissemination for MPEG-7 Based Digital Libraries. In *Research and Advanced Technology for Digital Libraries*, pages 256–268. 2004.

[Win85] G. Winskel. On Powerdomains and Modality. *Theoretical Computer Science*, 36:127–137, 1985.

List of Figures

List of Tables

List of Algorithms

Curriculum Vitae

For protection of privacy, the electronic version of this thesis does not contain the curriculum vitae of the author.